CHAPTER 1

Green ICT: History, Agenda, and Challenges Ahead

Mohammad Dastbaz Leeds Beckett University, Leeds, UK

Keywords

Industrial revolution

Information communications technologies

Internet

Integrated circuit

Sustainable ICT

Big data

Cloud computing

Introduction

The dawn of the Industrial Revolution and the growth of machine-based industries changed the face of our planet for good. While Charles Dickens darkly depicted poverty, disease-ridden cities, and squalid living conditions in *Great Expectations*, the replacement of the farming/cottage-type production industry with large factories changed Britain's skylines in the 18th century and signalled the beginnings of monumental change and innovation in addition to immense scientific discoveries.

The Industrial Revolution also fundamentally changed Earth's ecology and humans' relationship with their environment. One of the most immediate and drastic repercussions of the Industrial Revolution was the explosive growth of the world's population. According to Eric McLamb (2011), with the dawn of the Industrial Revolution in the mid-1700s, the world's population grew by about 57% to 700 million, would reach 1 billion in 1800, and within another 100 years, it would finally grow to around 1.6 billion. A hundred years later, the human population would surpass the 6 billion mark. This phenomenal growth in population put enormous pressure on the planet, forcing it to cope with a continuously expanding deficit of resources.

The transformation from cottage industry and agricultural production to mass factory-based production led to the depletion of certain natural resources, large-scale deforestation, depletion of gas and oil reserves, and the ever-growing problem of carbon emissions—mainly the result of our reckless use of fossil fuels and secondary products. The pollution problem following the Industrial Revolution led to the atmospheric damage of our planet's ozone layer as well as air, land, and water pollution.

The two world wars in the early twentieth century brought with them catastrophic human and natural disasters as well as rapid development of military technologies. These developments laid the groundwork for the emergence of what some would like to call the Second Industrial Revolution.

The Second Industrial Revolution—The Emergence of Information and Communication Technologies

In a visionary paper entitled “As We May Think” (published in *Atlantic Monthly* July 1945), Vannevar Bush, the scientific adviser to President Theodore Roosevelt, predicted a “bold” and exciting future world, based on “memory extended” machines. He wrote:

Consider a future device for individual use, which is a sort of mechanized private file and library. It needs a name, and, to coin one at random, "memex" will do. A memex is a device in which an individual stores all his books, records, and communications, and which is mechanized so that it may be consulted with exceeding speed and flexibility. It is an enlarged intimate supplement to his memory…

It was not long after this that the first large-scale technological innovations started to appear, marking the beginning of a new dawn in information and communication technologies, changing almost every aspect of our lives.

In 1946, the first new generation of computers emerged from US military research. Financed by the United States Army, Ordnance Corps, and Research and Development Command, the Electronic Numerical Integrator and Computer (ENIAC) was announced and nicknamed the “giant brain” by the press. In reality, ENIAC, as compared to the smartphones of today, had very limited functionality and capabilities. According to Martin Weik (December 1955), ENIAC contained 17,468 vacuum tubes, 7200 crystal diodes, 1500 relays, 70,000 resistors, 10,000 capacitors, and around 5 million hand-soldered joints. It weighed more than 27 tons, was roughly 8 × 3 × 100 ft (2.4 m × 0.9 m × 30 m), took up 1800 ft^2 (167 m^2), and consumed 150 kW of power. This led to the rumor that whenever the computer was switched on, the lights in Philadelphia dimmed.

In fact, all ENIAC could do was to "discriminate the sign of a number, compare quantities for equality, add, subtract, multiply, divide, and extract square roots. ENIAC stored a maximum of twenty 10-digit decimal numbers. Its accumulators combined the functions of an adding machine and storage unit. No central memory unit existed *per se*. Storage was localized within the functioning units of the computer" (Weik, 1961).

The Integrated Circuit (IC) Revolution

Although there was much excitement about the development of ENIAC, its sheer size and physical requirements, coupled with the limited functionality it offered for such a huge cost, used such systems limited to large mainly military-based research labs. At the same time that ENIAC was being developed and launched, there were other significant developments in the field of information technology.

Werner Jacobi, a German scientist working with Siemens AG, fielded a patent for developing cheap hearing aids based on an "integrated-circuit-like semiconductor amplifying device." Further development on a similar idea was pioneered by Geoffrey Drummer, working for the British Royal Rader. Drummer used the symposium on Progress in Quality Electronic Components in 1952 to discuss his revolutionary idea but was unfortunately unsuccessful in realizing his vision and building such a circuit.

It was not until 1958 that a young scientist working for Texas Instruments, Jack Kilby, came up with a solution for a truly IC and was able to successfully demonstrate a working version of it on September 12, 1958. The idea of being able to create much smaller systems, coupled with computer applications developed to offer greater functionality, soon changed the face of computing and led to the march of "machines." Jack Kilby won the 2000 Nobel Prize in Physics for his part in the invention of the IC, and his revolutionary work on the development of IC was named as an IEEE Milestone in 2009.

New Age of Computer Technology

According to the Historical Museum of computers online, there were only around 250 computers in use in the world in the 1950s. The development of IC and the microprocessor manufacturing industry and the emergence of corporations such as Intel meant that the world of computer technology had changed significantly. Intel's founder, Gordon Moore, observed that as the hardware industry grew rapidly, the number of transistors in a dense IC doubled approximately every two years—this estimation proved to be accurate and by the 1980s, more than 1 million computers were in use throughout the world. According to Gartner Inc., the number of installed PCs worldwide has surpassed 1 billion units, and it is estimated that the worldwide installed base of PCs is growing just under 12% annually. At this pace, it has been estimated that the number of installed PCs will surpass 2 billion units by early 2014. To this staggering figure, one must add the number of tablet and smartphone devices to understand the scale of this new technological age and its potential environmental cost.

It is important to note that it is not just the scale of the hardware development and emergence of more and more powerful PCs that have changed our way of lives but also the phenomenal development of applications available on our PCs and mobile devices.

I recall that in early 1992, I had my first taste of using the Internet and managed to log in to the Library of Congress information page, which showed me its opening and closing times. This was incredibly impressive, and I was thrilled that I could connect to the other side of the Atlantic at such speed; nevertheless, sitting in Kingston University's research lab, I also wondered how useful this information was for me if I could not access the rich content that the library had to offer. But it was only a matter of time before this changed, and the doors were opened to the “information super highway.” The clumsy text-based interface that I had used was replaced by a much more interesting and exciting Mosaic (graphic capable) interface and then the wonderful World Wide Web (WWW) arrived, and the rest, as they say, is history.

I do not think many of us living through the rapid development of this exciting yet unknown technology could have imagined that in a couple of decades our lives would be so dominated by it with frightening and somewhat crippling effects.

In reality, we have reached a stage where estimating the size and growth of the Internet and the WWW is extremely difficult, if not impossible. Nevertheless, one can find some terrifying figures online: according to www.factshunt.com, the size of the Internet and the WWW is something like:

• 48 billion—webpages indexed by Google Inc.

• 14 billion—webpages indexed by Microsoft's Bing

• 672 exabytes—672,000,000,000 gigabytes of accessible data

• 43,639 petabytes—total worldwide Internet traffic (2013)

• More than 900,000 servers—owned by Google Inc. (largest in the world)

• More than 1 yottabytes—total data stored on the Internet (includes almost everything; 1 yotta = 1000^8)

Global Mobile Computing and Its Environmental Impact

Clearly, the next question one has to ask is what the overall cost and the energy footprint of the new digital age are. Although there have been a number of attempts to identify a definitive cost model for the new digital age, clearly the scale and complexity of such calculations and the constantly changing data set have meant that at best we can come up with are intelligent guesses rather than definitive answers.

To answer the question of what the energy footprint of the new technology age requires, we must consider the electricity consumed by the world's laptops, desktops, tablets, smartphones, servers, routers, and other networking equipment (Gills, 2011). The energy required to manufacture these machines also needs to be included, as more importantly, does the energy required to keep such a wide range of devices and systems operational (such as the fact that smartphones typically require charging on daily basis).

Another key issue considered by researchers is the enormous volume of data transferred across our global mobile network of devices and the cost associated with maintaining and transferring these data.

Coroama et al. (2013), discussing the "electricity intensity of the Internet and its data consumption," point out that studies have already explored the electricity intensity of Internet data transfers (such as Koomey et al., 2004; Baliga et al., 2007, 2008, 2009; Taylor and Koomey, 2008; Weber et al., 2010; Hinton et al., 2011; Kilper et al., 2011; Lanzisera et al., 2012). Comparing their estimates is difficult because of inconsistent boundaries, data uncertainties, and the range of methodologies used.

In an interesting report sponsored by the National Mining Association American Coalition for Clean Coal Electricity) produced in August 2013, Mark Mills (CEO of Digital Power) states, "The information economy is a blue whale economy with its energy uses mostly out of sight. Based on a mid-range estimate, the world's Information Communications Technologies (ICT) ecosystem uses about 1500 TWh of electricity annually, equal to all the electric generation of Japan and Germany combined as much electricity as was used for global illumination in 1985. The ICT ecosystem now approaches 10% of world electricity generation.

Or in other energy terms the zettabyte (1000^7) era already uses about 50% more energy than global aviation.”

Referring briefly to the beginning of this chapter, human society's rapid industrial development over the past three centuries has caused far more lasting damage to our environment than ever before. As we destroy our forests and plunder our natural resources, the promise of technological innovations in saving our planet from further damage has not yet been realized. The ever-growing demand for data as a commodity that rules our lives along with the concept of “never switching off” and “contact on demand” with the requirement to keep the devices necessary to entertain this “new digital age” has brought with it further serious challenges that require immediate attention. Cloud computing was an incredibly promising concept that was perhaps hyped beyond reasonable measures to convince mass consumer migration. The matter of cost was never discussed or scrutinized as a factor before multinational corporations “migrated” us into the cloud. Although the move toward cloud computing has provided us with new opportunities that cannot be overlooked, the era of “big data” and the computing power required to deal with them both in terms of the data's energy consumption and technical complexity is one of the key areas of urgent further research and development. The US Environmental Protection Agency estimates that centralized computing infrastructures (data centers) currently use 7 GW of electricity during peak periods. This translates to about 61 billion kilowatt hours of electricity used. As already indicated, we are moving to the age of zetta and yotta data (1000^7 and 1000^8), but pressure from environmentally conscious consumers has forced computer and Internet giants such as Microsoft, Google, and Yahoo to build their new data centers on the Columbia River, where there is access to both hydroelectric power and a ready-made source of cooling.

The Agenda and Challenges Ahead

So what are the key agenda items for the “Green Information and Communication Technology” debate, and what are the key challenges facing researchers and developers in this area?

• While there is focus on big infrastructure and big data computing, the debate quite often tends to overlook the large number of existing “legacy” systems (which are neither green nor efficient). Even our current laptops are viewed as “old legacy” systems. While the statistics show that there are around 2 billion PCs in operation currently, most of these systems suffer from “old” hardware (i.e., power hungry) designs. It is not surprising that manufacturers such as Intel have spent billions of dollars designing the next generation of microprocessors (“Haswell”) and moving toward fanless, less power-hungry systems.

• Another key challenge is how we measure ICT performance and sustainability and what tools we can use to provide reliable data. It is clear that unless we have a reliable methodology to measure ICT suitability, we will be experiencing what environmental campaigners have been experiencing over the past decades: claims and counterclaims and data being dismissed for not being accurate or substantial enough to corroborate our arguments.

• One of the most critical challenges facing green Information Technology is the legal framework within which system developers and providers need to work. Although recently the European Union (January 2015) brought in regulation to European Union rules to oblige new devices such as modems and internet-connected televisions to switch themselves off when not in use, nevertheless, we are far from having robust sets of enforceable regulations for green IT on the national or international level.

• While cloud computing was hailed as the "green way" of moving away from device-dependent clunky power-hungry applications and data storage approaches, and although it could be claimed that the use of virtualized resources can save energy, a typical cloud data center still consumes an enormous amount of energy. Also, because the cloud is comprised of many hardware and software elements placed in a distributed fashion, it is very difficult to precisely identify one area of energy optimization.

• Some of the most interesting areas of research in green Information Technology concern "energy harvesting" or "energy scavenging," and the concept of the internet of things (IoT). Energy harvesting is exploring how we can take advantage of various ambient power sources scattered all around us. The concept of the internet of things examines a series of technologies that enable machine-to-machine communication and machine-to-human interaction via internet protocols. Being able to use the World Wide Web and the global connectivity of billions of machines, smartphones, and tablets to monitor and control energy usage can have a tremendous impact on developing a much more environmentally friendly computing world that is sustainable.

To predict the future and particularly the future of green Information Technology is a futile exercise that one should not entertain. The pace of changes over the past two decades has proven to us that even the most visionary thinkers of our time struggled to provide us with a vision and a road map to follow. Perhaps the best way to conclude these introductory remarks is quote the visionary who gave us the World Wide Web and these days is very worried about how large multinational corporations and indiscriminate government surveillance can ruin a tool that has provided such amazing opportunities for humankind. In the words of Sir Tim Berners-Lee, "One way to think about the magnitude of the changes to come is to think about how you went about your business before powerful

Web search engines. You probably wouldn't have imagined that a world of answers would be available to you in under a second. The next set of advances will have a different effect, but similar in magnitude.”

References

Baliga J, Ayre R, Hinton K, Sorin WV, Tucker RS. Energy consumption in optical IP networks. *J. Lightwave Technol.* 2009;27(13):2391–2403.

Baliga J, Ayre R, Sorin WV, Hinton K, Tucker RS. *Energy consumption in access networks. In: Optical Fiber Communication Conference and The National Fiber Optic Engineers Conference (OFC/NFOEC).* San Diego, CA, USA: Optical Society of America; 2008.

Baliga J, Hinton K, Tucker RS. Energy consumption of the internet. *Paper presented at Joint International Conferences on Optical Internet, and the 32nd Australian Conference on Optical Fibre Technology, COIN-ACOFT.* Melbourne, VIC, Australia: 24–27 June; 2007.

Berners-Lee, T., 2015. BrainyQuote.com. Xplore Inc. http://www.brainyquote.com/quotes/quotes/t/timberners444499.html (accessed 02.02.15.). Read more at http://www.brainyquote.com/citation/quotes/quotes/t/timberners444499.html#cwclkW9U8U6sRqvV.99.

Bush V. As we may think. *Atlantic Monthly J.* July 1945. http://www.theatlantic.com/magazine/archive/1945/07/as-we-may-think/303881/?single_page=true.

Coroama V, Hilty LM, Heiri E, Horn F. The direct energy demand of internet data flows. *J. Ind. Ecol.* 2013;17(5):680–688. doi:10.1111/jiec 12048.

Gills, J., 2011. Internet responsible for 2 per cent of global energy usage. New Scientist http://www.newscientist.com/blogs/onepercent/2011/10/307-gw-the-maximum-energy-the.html.

Hinton K, Baliga J, Feng M, Ayre R, Tucker RS. Power consumption and energy efficiency in the internet. *IEEE Netw.* 2011;25(2):6–12.

Jacobi, W/SIEMENS AG, 1952. “Halbleiterverstärker” priority filing on 14 April 1949, published on 15 May 1952.

Kilper DC, Atkinson G, Korotky SK, Goyal S, Vetter P, Suvakovic D, Blume O. Power trends in communication networks. *IEEE J. Sel. Top. Quant. Electron.* 2011;17(2):275–284.

Koomey J, Chong H, Loh W, Nordman B, Blazek M. Network electricity use associated with wireless personal digital assistants. *J. Infrastruct. Syst.* 2004;10(3):131–137.

Lanzisera S, Nordman B, Brown RE. Data network equipment energy use and savings potential in buildings. *Energy Efficiency.* 2012;5(2):149–162.

McLamb, E., 2011. The Ecological Impact of the Industrial Revolution Ecology Global Network | News and Information for Planet Earth. http://www.ecology.com/2011/09/18/ecological-impact-industrial-revolution/.

Taylor, C., Koomey, J., 2008. Estimating Energy Use and Greenhouse Gas Emissions of Internet Advertising. http://energy.lbl.gov/EA/emills/commentary/docs/carbonemissions.pdf.

Weber CL, Koomey JG, Matthews HS. The energy and climate change implications of different music delivery methods. *J. Ind. Ecol.* 2010;14(5):754–769.

Weik, M.H., December 1955. Ballistic Research Laboratories Report No 971 - A Survey of Domestic Electronic Digital Computing Systems - page 41. US Department of Commerce. Retrieved 2009-04-16.

Weik MH. *Ordnance Ballistic Research Laboratories.* MD: Aberdeen Proving Ground; 1961.

CHAPTER 2

Emerging Technologies and Their Environmental Impact

Colin Pattinson Leeds Beckett University, Leeds, UK

Abstract

Innovation has been a characteristic of human development from our earliest stages. Innovation in technology has allowed, and continues to offer, opportunities to change our way of life, often in ways which were not predicted by contemporary observers. Some developments are totally new: many are based on new uses for the technology which already exists. This chapter considers the interrelationship between growth of utilisation, development and obsolescence in a number of scenarios: video conferencing; dematerialization; travel advice and building management including intelligent metering. It explores the benefits and drawbacks of increased use of technology, and concludes by offering optimistic and pessimistic views of the future.

Keywords

Environmental impact

Moore's law

Backward compatibility

obsolescence

Dematerialization

Resource saving

Building management system

Introduction

That we live in an age of rapid technical development is, if anything, a massive understatement of the actual situation. It is remarkable that so many of these developments have resulted in devices that have become almost indispensable to everyday activity; it is impossible to envisage life without things such as mobile phones, digital TV on demand, or computer programs and applications such as Google, eBay, and Facebook. It is also remarkable that these technologies have become such an integral part of everyday life (at least in "developed nations") over such a short period. In spite of (or maybe because of) the rapid pace of development during the past 20 years, there seems to be no letup in the expected pace of future innovation: technologies such as 3-D printing are only beginning to be exploited, and devices such as wearable computers are beginning to emerge. Even without these newly emerging uses of existing technology and of the radical new technologies that are yet to take shape, we are in the grip of one very clearly predictable consequence of technological advancement: the fact that more users are making more use of more devices to do more things. In addition, it is usually the case that each of these new activities requires more resources as it becomes more complex.

The result of the connection between easier (cheaper, more rapid) production and a wider range of uses for the product has been seen before in many situations: eighteenth-century improvements in iron manufacture led not to making the same limited set of products in a shorter time using fewer resources but to a much wider range of iron products. The mechanization of textile (wool and cotton cloth) manufacture in nineteenth-century Great Britain meant that more cloth was available to be made into a wider range of clothing. Mass production techniques such as those associated with Henry Ford led to producing more cars rather than building the same number of vehicles as before but at lower cost, and more quickly (in days rather than weeks). The most commonly quoted example of this phenomenon is the *Jevons paradox* (Alcott, 2005). The Victorian economic scientist W.S. Jevons noted that improvements in the efficiency of the steam engine had led not to the same amount of work being done with fewer machines but instead to an increased number of steam engines accompanied by a growth in the scale and scope of their application. He could have been predicting the development of information technology (IT) and computing power. Instead of running the same set of application software in an efficient manner, the availability of more processing power, greater data storage, and quicker, more reliable data transfer have allowed the creation of a vast range of applications including the examples mentioned.

A full discussion of capitalism and market economics is not appropriate here, but suffice it to note that the cost of investing in improved production techniques demands that the production system make more units of product and sell them at the demanded profit and, in turn, that new outlets and uses be found for them.

In some of these cases, the improvement process is continuous, interrupted only by sudden changes (the replacement of human labor by robots in much car manufacturing is one such example). In others, new products overtake the old: many former cast metal products are now made in plastic (children's toys and garden furniture being two such cases). Sometimes the development process itself reaches a point at which no significant further development is possible. In the domain of IT, the most obvious case is that of Moore's law. There are physical limits to the miniaturization of the transistor-based integrated circuit devices governed by immutable characteristics such as the properties of light and the size of the electron. However, this does not mean that development is likely to stop the demand for new technologies driven by the applications and services of those technologies but seems set to continue beyond the limit of Moore's law—even without the prospect of many more years of improvement in silicon-based IT and even if new developments such as quantum computing fail to materialize.

The number of devices provides an additional multiplying factor. In common with many “developed” countries, some years ago the United Kingdom passed the mark of having more registered mobile phones than it has citizens. There is similar growth worldwide, not by any means limited to “first-world” countries. The billionth PC was shipped in 2008, and the data—in its many different formats—that this growing collection of devices generates and processes also grows year on year. The volume of stored data is actually growing more quickly as the twin factors of easier production and greater precision make it simpler to produce more while the declining cost per unit of storage reduces the need to be selective about what is kept.

The final factor to be considered is obsolescence. In some cases, obsolescence is physical, caused as technology “wears out”: each time an on/off switch is used, wear (metal fatigue) occurs; memory read/write operations can be carried out reliably only a limited number of times before the magnetic characteristics of the device become unreliable. In other cases, obsolescence is brought about by external factors: the UK government's decision to reallocate the spectrum formerly used by analog TV transmission meant that TV sets and associated recording devices designed to receive analog transmission no longer have a signal to receive. In other situations, obsolescence arises as the result of newer devices that are more desirable because either they possess greater functionality or they are perceived as fashion items. Smartphones offer examples of both these cases: some are replaced because a newer version offers more or better quality user experience, others because they are no longer the current “must-have” devices. Lastly, devices become unusable through damage and are replaced by newer ones.

The purpose of this chapter is to show that all these changes tend to increase the amount of computing processing and memory needed and that delivering that processing and managing that memory lead to an increase in energy requirements. It seems likely that this increase may far outstrip any “savings” made in efficiency improvements *within the technology itself*: therefore, if we are to maintain or improve the overall environmental balance sheet for emerging technologies, those technologies must deliver significant positive environmental benefits in their own right. We will therefore conclude by considering these benefits and placing them into context.

Number of Connected Devices

The continued development of existing technologies, as opposed to the emergence of new ones, is likely to provide the majority of growth in the number of devices, although there are likely to be new technologies (wearable computers, the “internet” of things) that will also have significant impact on this growth.

As mentioned in the introduction to this chapter, the growth in Internet-connected devices is clear and some of the numbers that capture this bear repetition:

• Of the world's population, 50% possesses a mobile phone (World Bank, 2014).

• In many countries, there is already more than one registered mobile phone per capita (Stonington and Wong, 2011) although the data do not show how many of these remain active, and it is likely that a significant proportion is unused, having been superseded but not disposed of.

• By the end of 2013, 1.78 billion PCs and 1.82 billion units of browser-equipped mobile devices were in use (Gartner, 2013).

• In the United Kingdom in 2012, there were an estimated 10 million office PCs, and it was estimated that 50% of the working population used a PC in their daily work; this is expected to increase to 70% by 2020 (1E, n.d.).

• The number of smartphones in use is predicted to pass the number of PCs in use during 2014 (Blodget, 2013).

• *Sales* of tablet devices are likely to exceed those of PCs during 2015 (Blodget).

• By 2020, there are predicted to be 50 billion Internet-connected devices; much of this will be a consequence of the Internet of things explored in subsequent paragraphs, but almost all of the devices (phones, PCs, and tablets) mentioned possess some form of Internet connectivity (Chiu et al., 2010).

Most of these devices and most of the emerging applications are not stand alone; they rely on technologies such as cloud computing, device connectivity, and information sharing to deliver their functionality. This makes it pertinent to include the growth in networking and storage within the consideration of the environmental impact of emerging technologies.

The increase in the numbers of devices, the features each one possesses, and inbuilt complexity all tend to increase the volume of data. In addition to the obvious fact that more devices are likely to produce more data, registering, tracking, interconnecting, monitoring, and securing large numbers of devices are all data-generating activities. Additional functionality also tends to result in more data: a one-color document (on screen or printed) is a rare sight in most offices; people who have digital cameras in their smartphones use them to capture events that would have gone unrecorded by the previous generation of analog camera users. The widespread adoption of cloud storage means that the need to be selective about which data to retain—a need already reduced by growth of on-board storage capacity—has now been pushed even further into the distance. Clouds offer almost limitless data storage at low additional cost, so there is little incentive for “housekeeping” to remove redundant, duplicate, or out-of-date information. The inevitable result of all this has been an increase in the overall volume of stored data: in 2011, it was estimated that a zettabyte (10^{21} bytes) of digital data existed in storage systems (Wendt, 2011); by the time this book is completed, it is expected that this will have quadrupled.

Consideration of data storage leads us to data centers, whose power consumption doubled over the four years between 2007 and 2011; in the single year of 2012 (a year of major investment in cloud technologies), this increased by an additional 63% (Venkataraman, 2013). Note that these figures relate to power consumption, not the actual number of data centers or the units installed within them. If we accept that efficiency has improved over that time, we are forced to conclude that the power consumption is an underestimate of the installed units because each unit should require less power in 2012 than its equivalent in 2007. Since then, growth has continued, albeit at a lower rate, but still in excess of 10% per annum. Not surprisingly in view of this, data centers are responsible for a significant portion of the total emissions for the IT sector (14% of the total in 2007). It is estimated that this will rise to 18% by 2020.

Greenpeace's 2011 estimate (Greenpeace, 2013) that, considered as a country, “the Internet” would be in fifth place, ahead of Russia and below Japan in a report of electricity use, is a compelling illustration of the energy consumption required to support the world's demand for IT. The future seems likely to be less one of new technologies and more of the ever-increasing use of existing technologies to create more data. Cisco has dubbed the five years ending in 2018 as “the zettabyte era,” a recognition of the fact that global networks will carry that amount of traffic in the calendar year 2016, increasing to 1.6 ZB in 2018 (Cisco, 2014). In light of this, it is interesting to estimate the likely electricity requirement of the global telecommunications system of 2016.

In a study of conventional wired communications technology (mixed fiber and copper) Coroama et al. (2013) offer a “pessimistic” estimate that overall Internet *transmission* uses 0.2 kWh of electricity per gigabyte (GB) of data. “Pessimistic” means that the estimate is a worst-case scenario, and improvements in energy efficiency will probably reduce the estimated use over time. However, even if we assume that this pessimism has doubled the actual figure to move 1 ZB at 0.1 kWh/GB will require 100 TWh of electricity per year. An estimate by Raghavan and Ma (2014) uses a different definition of the “Internet” that includes end devices and a full life cycle costing (both excluded by Cororama et al. whose focus is solely on the communication links). As a result, their estimate is significantly higher at 8 kWh/GB; applying this figure to Cisco's “zettabyte Internet” would mean an energy requirement a factor of 100 higher than the preceding figure. For comparison, the total domestic energy consumption of the UK for 2010 was 112,856 Wh/GB (Rose and Rouse, 2012).

Additional growth in what might be considered *hidden connectivity* will occur. The internet of things is a concept in which almost any object can or will possess Internet capability. Much of this could create very positive forces for good by delivering reductions in energy use without requiring operator (human) intervention. Obvious examples are the washing machine, which is able to determine the optimum combination of time, heat, and water use for its current load or the in-vehicle systems that will provide the most fuel-efficient route and driving patterns for current road and weather conditions. The very need to provide such connectivity means that each device effectively becomes a computer in its own right. Providing devices in this quantity will require the creation of more IT with consequential increases in the demand for raw materials and the energy needed in operation as well as the need for disposal. Allowing these devices to communicate with each other, whether wired or wirelessly, will make a significant contribution to the growth in traffic predicted by Cisco.

Increased Functionality

Added functionality is widely used as a selling point for new products. The development and upgrade cycle of consumer electronic products is the most apparent illustration, for example, the way in which improvements in camera resolution, creating higher-quality images, have been a selling point for new devices. A higher resolution camera will generate more data (each pixel is 3 bits, so a 12 megapixel camera at the lowest end of the current market requires 36 Mb of uncompressed data per image before any compression. Higher resolution calls for one or more pixels, information per pixel, and less compression).

Another rapidly growing consumer product is in-vehicle navigation. Here the same factors are apparent. New units are marketed on their performance, accuracy, and reliability: a more precise geolocation system will require more processing to calculate with greater precision (the basic in-vehicle geolocation systems typically transfer less than 2.5 MB per day in constant use and offer a positioning accuracy of 10-15 m, higher levels of precision, an accuracy call for increased sampling rates, and more processing, for example, the use of "least squares" to minimize misfits between modeled and real location [Bilich, 2006]).

In both cases the result is the same: the amount of resource (energy or hardware) is larger: more storage and transfer capacity are required because the volume of data generated is larger, and/or there is more processing of this additional data to provide the higher levels of accuracy and precision.

Increased Number of Separate Functions

Multifunction devices (MFDs) are now commonplace, whether in the guise of the phone that also acts as a camera, personal organizer, Internet access device, entertainment center, mapping and location system, and so on, or the office printer that also provides copying, scanning, and image processing. On public transport, a single device can issue and check tickets, provide a live timetable and journey planning information, ATM functionality, and the generation of data relating to cash sales. Less obviously, single devices provide multiple functions in networking, connecting both wired and wireless devices, and a single vehicle system can support both engine maintenance and management and route control. Initiatives in the field of wearable computing expand this use with devices that allow health monitoring to be added to the existing functions of the mobile phone.

This could be viewed as a good thing because it should reduce the number of separate devices needed by any one person, but the reality is rather less distinct. We argue that the increased number of functions also contributes to the negative environmental impact of technology for the following reasons.

• *A proportion (often significant) of this added functionality is unnecessary*: Often many of the added functions available are rarely used if at al, but they still require support, which is rarely energy free. Simply running a background process on a device consumes CPU cycles and hence power. Unless the user chooses otherwise, it is quite likely that the application will be monitored and upgraded, regardless of whether it is actually used.

• *Unsuitability for multiple purposes*: It is often difficult to combine the demands of different user interfaces into the design of a single device while retaining the comfortable and efficient (in human terms) use of any one function. This can often lead to compromise, resulting in dissatisfaction. Although it might be almost acceptable to take holiday photographs with a regular tablet computer, using it as a “normal” telephone (without the addition of a hands-free earphone and microphone) would not be acceptable to most people. How this dissatisfaction is then manifest may have a further environmental impact: the user may discard the device in an attempt to find a “better” multifunctional device or purchase more than one device, using each for the subset of roles the user finds most satisfactory (and then adding to the communication network load by synchronizing them); or other technology may be added in an attempt to make the use more comfortable. As an example of the latter case, consider the growing market in detachable keyboards for tablet computers to meet the need for a more “user-friendly” keyboard for sustained operator.

• *Separation of ownership/primary use*: A rush hour journey on public transport will quickly reveal that many users have more than one example of what is ostensibly the “same” device. This typically is one mobile phone for work and one for personal use. This practice may be driven by individuals' wish to maintain a separation between their public and private lives or by security-driven management policies to avoid misuse of company-provided equipment.

Increased Demand for Speed and Reliability

Describing the communications between units of the British Army in South Africa during the military campaigns of 1899-1902, Arthur Conan Doyle gives us what was then clearly considered a remarkable example of speedy and reliable communication: “it is worthy of record that … at a distance of thirty miles [48 km], they succeeded in preserving a telephonic connection, seventeen minutes being the average time taken over question and reply” (Doyle, 1902). In the 112 years since then, military (and more peaceful) operations in both sound and vision are controlled remotely over thousands of kilometers in real time.

The use of the word *controlled* in the modern-day description is in deliberate contrast to *question and reply* in the 1902 example: whereas local commanders in the army of 1902 would be expected to make local decisions based on information received, the *remote control* of an operation now calls for instructions to be issued and received with 100% reliability and accuracy.

Emerging trends in a number of fields are the requirements for increases in the use of such systems. Smart city initiatives call for monitoring environmental characteristics, traffic patterns, power, water and sewage flows, traffic systems, water and power grids, and building heating and ventilation units. The mapping and location requirements of navigation systems (in vehicle or handheld) that will guide users to specific locations will need real-time information about current location, and automated transport (exemplified by the recent announcements of driverless cars) will call for ever more precise and accurate location. At the moment, there is no suggestion that driverless cars be remotely controlled, although many mass transit systems do operate in a highly automated mode (e.g., the Docklands Light Railway in London), and this will be extended whether through entirely new systems or additions or modifications to existing ones. In those applications in which remote control of systems whose incorrect functioning can present a risk to life and property, the speed and reliability of information and commands are paramount.

Similar demand for speed, reliability, and accuracy now exist in commonplace activities. Although they may not be essential, these device have become desirable or expected. The consequence of this is the need for systems that deliver these three requirements. In turn, these demands are met by the user of higher performance devices that typically require more memory and processing power. The standard approach to providing increased reliability is through *redundancy* with stand-by units both hot (ready to go instantly) or cold (able to be powered up at short notice); by load sharing to avoid overloading any one component; and by system designs that seek to eliminate any single point of failure. It hardly needs to be stated that the overall result of attempts to meet the demands for speed, reliability, and accuracy in any single activity is often to increase the volume of resource needed to support that activity.

Obsolescence—The Problem of Backward Compatibility

The IT industry is one in which change and innovation are rapid and often fundamental. Consider the change from mainframe computers to minicomputers to stand-alone PCs and then to networked PCs, the client-server era, wireless networks, and the current trend to use cloud and mobile devices. These step changes—occurring roughly once per decade—have meant that otherwise fully operational and functional systems have been rendered obsolete before their designed lifespan has expired. *Backward compatibility*—allowing newer and older software and equipment to work together—may be achievable for a time but is not sustainable in the longer term without constricting development. The need to support a variety of different hardware types (old and new) also adds to the complexity of the software development process for applications, operating systems, and device drivers. After some time, the cost and complexity of maintaining backward compatibility will become too high and will be dropped. The majority of users typically will have already moved on to the newer product as part of normal update/renewal processes. Those who are left then must decide whether to continue with (now unsupported) equipment or to seek alternative applications that will run on their current equipment and are supported by the developer. Note, however, that this approach brings the added cost of learning the new system and possibly the need to convert data formats.

The speed of development and upgrading of technology attenuates this process, giving rise to the well-known situation of fully functional equipment being discarded because of obsolescence rather than being worn out.

Within the PC environment, Microsoft's recent decision to cease support for the XP operating system is a classic example of a step change, and user response is a classic example of what happens as a consequence (Covert, 2014). Although it is possible to install other operating systems, it is likely that many of the current XP devices will go out of use in relatively short order as application support, development, and (probably more significantly) security updating are discontinued. The difficulty—real or perceived—of transiting to a newer operating system will add to the likelihood of accelerated obsolescence.

Another example of accelerated obsolescence, already mentioned in the introduction to this chapter, is the UK's decision to no longer provide analog TV signals. Here the driving force was government policy rather than manufacturer decisions. Although newer equipment was already being adopted by a significant proportion of users as a consequence of demand (or desire) for the additional features offered by cable and satellite provision, it is clear that the government's decision to reallocate the frequency bands used by analog TV in order to auction them to mobile operators created a step change in this process. The provision of set-top converter units offered the option to retain existing analog units, but the added functionality of new TV sets—particularly HD—has led to an increase in sales of the newer ones. Whether the sets rendered redundant by this are stored or disposed of is unclear.

The Other Side of the Balance Sheet—Positive Environmental Impacts or the “Other 90%”

We have already seen that the likely result of continued development in the use, range, and application of IT will be an increased demand for the resources required to create, maintain, and dispose of the various hardware items in use. The use of *resources* here refers to everything from the raw materials (including rare earth metals) and other chemicals (acids for etching, water for cleaning) and the energy used in the manufacturing process to electricity for powering the devices and the energy needed to handle them at the end of their lives. A wider definition would also include factors such as the energy required to transport both new and discarded raw materials and finished products.

The case is frequently made that much of the focus has been on the greening *of* IT systems; that is, to make the technology more energy efficient, driving down the energy consumption of data centers, and so on. However, if we take the widely quoted figure that IT is responsible for 10% of the world's energy consumption, this leaves the question of the “other 90%.” In addressing this, we consider the opportunities for greening *by* IT (i.e., using the technology to reduce the energy use of other aspects of human activity). Most prominent among these opportunities are transport and building heating and lighting. They are simple because they are the largest consumers of energy and therefore even a small proportional reduction in them would be significant.

One possible approach to create a balance sheet is to consider a particular activity that is performed in a particular way that allows a calculation of the resource required to support that activity. Then consider how the activity would be undertaken using IT as an enabler, calculate the resource requirements for *that* mode of operation, and compare the two. One activity—replacing the intercontinental business trip by videoconference—will become almost the de facto comparative factor.

Constructing a balance sheet for an activity such as a trip involves determining the energy used by a single air traveler on a return journey to conduct a face-to-face meeting and then working out the energy required to support a videoconference of the same time length. Such a calculation makes a number of assumptions, including that a videoconference is as productive as a face-to-face meeting and that removing a number of *passenger* trips would necessarily lower the number of *airplane* trips as opposed to the same number of planes traveling at reduced occupancy.

Attempts to calculate the actual energy (or resource) used for a particular transaction results in very widely variable outcomes; there are significant variations in the determination of which devices are considered to support a transaction. The problem is further complicated by the task of apportioning the resource use of any shared device that actually handles a specific transaction and by the discussion of whether the device in question would have used any less resource had the specific transaction not taken place (e.g., if a particular call did not take place, would a router that was not required to route the particular call have used any less energy or resource?).

Videoconference as an Alternative to Business Travel

Perhaps unsurprisingly, most calculations of this form consider the use stage alone; there is no attempt to include the embedded energy of manufacturing a passenger airliner and then apportioning a part of the embedded energy to a given passenger on a particular journey. Nor should it be noted whether these calculations include any consideration of the energy costs of building and operating airports or of the cost of operating the communications infrastructure necessary to direct air traffic. Even without these added embedded energy costs, the typical energy balance sheet is heavily in favor of the videoconference solution. For example, Raghavan and Ma (2014) calculate that replacing 25% of the current 1.8 billion air journeys by videoconference would “save about as much power as the entire Internet [consumes].” An added benefit is that the energy saved is energy that would be generated by burning aviation fuel.

Dematerialization of Product Chain

Perhaps a more interesting and exciting opportunity is offered by dematerialization of the supply chain for a particular product item. Consider the situation in which some mechanical device (e.g., an office air-conditioning unit) fails. Currently, the sequence of operations is typically as follows: a maintenance engineer is called to the site, diagnoses that the problem requires replacement of a particular component, which is then delivered to the site, and the engineer then returns to perform the repair. Alternatively, the engineer may obtain the component or, very rarely (or so it seems), has the required part on hand, but this is only at the cost of maintaining a stock of such parts, resulting in a larger vehicle to carry more weight. A 3-D printer on site would allow the replacement part to be produced locally and fitted in one visit. As an additional IT-related benefit, remote diagnostics (also supported by IT) would provide an additional enhancement: the engineer could arrive knowing that the task was to replace a particular component and find it waiting upon arrival, thus saving time.

The potential for 3-D printing is enormous. Although the spare part example is one of the more trivial ones, it is easy to envisage other products being “delivered” in the same way; examples include clothing, food, and buildings, both terrestrial (BBC, 2014) and extraterrestrial. NASA and the European Space Agency are currently considering the possibility of constructing a moon base using 3-D printers that use lunar soil as the raw material. The cost of transporting the 3-D printers would be significantly less than that of transporting building components from earth (NASA, 2014). Although it is still necessary for the raw materials to be available for the printer to use on site and the limitations of choice and quality (particularly for 3-D food) may leave something to be desired with current technology, 3-D printing could clearly offer potential energy savings in reducing the transportation and packaging of finished goods.

What are arguably more trivial examples of dematerialization are already with us: the decline in printed newspaper, magazine, and book volumes brought about by their replacement with e-readers is beginning to have a noticeable impact on sales and production methods. The change to online consumption of music and home entertainment has already led to significant declines in the supply of physical media such as CDs and DVDs with consequences in the decline of the high street outlets for these products.

The last example is only one part of an even more significant disruptive use of IT: the online shopping boom and its impact on shopping habits. We do not believe that there has been any attempt to calculate the cost (in energy and resources) of the typical online shopping spree in the run-up to Christmas 2014. Nor has there been an attempt to compare the cost with the energy and resources that would have been necessary to collect the same set of goods by traditional shopping. It seems likely, however, that savings quoted for videoconferencing coupled with the economies of scale and the logistical improvements achievable by the major online retailers and the opportunities for planning fuel-efficient delivery would occur for the online activity. If products such as clothing, toys, kitchen goods and even food items were to be 3-D printed, the cost savings would be even higher. Whether we are prepared to accept the societal impacts of this (the inevitable disappearance of actual shops in towns and cities) or the loss of the physical and mental activity of the shopping trip on us as humans adds extra complications to any such calculation.

Travel Advice/Road Traffic Control

Although the replacement of journeys and physical movement of goods described here offer the potential to decrease some travel, the human need (or desire) to move around in vehicles of some form apparently is likely to continue. Travel in more populates areas can be met by a mixture of private and public transport, and IT can make a positive difference to overall energy use. We already have implemented a number of these: modern automobiles are fitted with engine management systems and other devices to ensure optimum use of fuel; communications systems allow information on traffic flows and road conditions to be made available so that drivers can consider changing travel plans to avoid congestion. Public transport vehicles also benefit from such devices, and a well-planned information system allows control of the whole network while keeping customers informed and allowing them to plan their journeys. Finally, citywide traffic control systems provide traffic management across a city so that local changes to traffic flows, such as sequencing traffic lights and rerouting traffic, will reduce delay, thus offering the option to use less fuel. It is likely that these initiatives will further develop and expand to incorporate data such as weather forecasting (cold and rainy weather encourages the use of cars and buses rather than walking or cycling for essential journeys and reduces the number of nonessential journeys—both of which have predictable effects on transport patterns and demand).

Other uses of ICT in traffic and travel management are more subtle: charging fees that to persuade leisure users not to travel at peak times can be facilitated by tracking technologies such as automatic vehicle license plate recognition; requiring changes of public transport mode (e.g., bus to train to bus) is made easier by ticketing and using an integrated timetable.

Fuller integration of traffic management, journey planning, and timetable integration, together with longer-term planning and modeling of city infrastructure (to allow easier access to places of work and leisure), bring us to the reality of smart cities, the potential of which is now beginning to be exploited.

Intelligent Energy Metering

In the United Kingdom and other countries, most domestic energy use (gas and electricity) was until recently recorded and paid for using a labor-intense process. A “meter reader” would visit each household to physically make a reading (by reading a meter's dial and writing down the number on it). This reading then generates a bill, which is probably the first indication of their energy use that most customers have. On receipt of the bill, some users might take action to reduce use, but without a positive feedback loop and accurate (and timely) information, this is often a piecemeal activity. The location of meters in difficult to access locations was another disincentive to proactive energy awareness.

The concept of so-called smart meters can provide easier access to information about energy consumption in real time, allowing more obvious changes to be made to current energy consumption. A web-enabled meter can communicate real-time usage to an end user device (a smartphone or tablet) without requiring the user to locate and check the meter. Among other benefits, the time-consuming round of meter reading can be removed; if the data can be sent to the user's smartphone, it can also be communicated to the energy company.

Emerging applications of the potential for this technology will include energy suppliers' ability to negotiate and smart meters to balance demand, allowing customers to choose between suppliers using a number of criteria. Moving the negotiation processes further into the domestic arena, customers can determine the pattern of their energy costs for their household devices—especially washing machines and dishwashers—and can use them during periods of lower demand.

More controversially, there is the prospect that the energy supplier could control operation and behavior more directly by restricting supply. We are not aware of this actually taking place, but it is indicative of the challenges resulting from the opportunities for more control by more complex technology. The issue of the extent to which we are willing to allow technology to control our behavior and lifestyle is a topic worthy of a full research study in its own right but is beyond the scope of this book.

Building Management Systems

We are familiar with systems that control the temperature of buildings in which we live and work from domestic central heating controllers to office air-conditioning units. In large buildings, such as office blocks, hospitals, and university and colleges, these control systems are being merged with sensors and other input systems in the form of a building management system (BMS).

A complex BMS has the potential to have very detailed control and operation not just of the heating, ventilation, and air conditioning (HVAC) system but also lighting and security systems; they can also provide room ambience (mood music) appropriate to the occasion. By including inputs from weather forecasting systems, the BMS can adjust HVAC settings to prepare a building for coming weather conditions—for example, providing some heating while energy is at a cheap rate because the outside temperature is predicted to fall over a three- to five-day window rather than waiting until the temperature falls below a set level after two days of cooler weather, which requires using more expensive energy to provide required heating.

Other BMS-related opportunities include a linkage between room use and HVAC settings and modification to heat needs for different numbers of occupants or for different levels of activity; more active occupants would need less heat (or more cooling) with the less active the needing the opposite. Security could be enhanced via an intelligent BMS: movement detected in a room that is scheduled to be unoccupied would trigger an alarm; the ability to count the number of people entering a given room could be compared with the room's safe occupancy level, allowing action to be taken as necessary.

As with energy metering, it is likely that the potential range of uses to which this technology is put will be limited as much by human factors as by any limitations in the technology itself. The degree to which we are prepared to accept the “intrusion” that many of these applications bring with them—in both monitoring behavior and adjusting the environment—remains to be seen. Once data are available relating to an individual's presence in a given location (or to the intensity with which the individual is moving within that location), legal and regulatory action is needed to prevent the use of that information for other purposes—such as monitoring active working hours.

Saving IT Resources—A Drop in the Ocean?

It is clear that the effect of individuals making the changes that are available will have negligible (a drop in the ocean) impact on the overall energy consumption of IT—whether current or emerging. One person deciding to cull unwanted photographs stored “somewhere in the cloud” is not going to change the resource requirements of the cloud servers or the network that connects them. However, a number of individuals involved would probably be sufficient to have a measureable impact *if they all did the same thing*. The likelihood of this is, of course, increasingly small although workplace campaigns to raise awareness can have a limited but measureable effect. Campaigns to encourage office staff to turn off their computer systems on weekends typically have some impact: the problem is that the impact of the campaign and hence the level of response diminishes over a relatively short time scale. To make a major, lasting difference, legislators or providers need to take action; the most effective tool is financial—making it more expensive to produce and retain more data. Providers could also introduce a “delete by default” form of operation in which stored photographs are automatically deleted after a set time period unless the owner explicitly tags them for retention, thereby taking advantage of the inertia that affects the majority of people.

Another “drop in the ocean” response to the call for greater efficiency in the IT sector relates to the relative size of its contribution to the environment (10% of energy, 2% of greenhouse gas) compared to other sectors, not only the airline industry but also heavy manufacturing of such items as steel and chemical works. This is in addition to the underlying inefficiency of much fossil fuel electricity generation. Although this is undoubtedly true, a counterargument is that if some proportion of that 2% (or 10%) can be reduced, the opportunity should be taken. With energy demands at the level discussed earlier for the zettabyte network, even a small relative change is significant in absolute terms.

Conclusion

Our capacity for innovation and for acquiring (or coveting) innovative products has often seemed to be unlimited. Even the exhaustion of a particular resource often fails to have a major effect: we appear to be able to innovate to find a replacement either for the resource or the product that used it.

In the short to medium term, there is little reason to expect a reduction in the increased demands for IT products and services discussed in this chapter. Globalization means that those areas of the world that have yet to feel the benefit of increased IT will be more and more exposed to it. Market economics will cause more products to be made and create new needs to exploit them. Improvements in efficiency (through customer demand, energy cost, and legislation) will continue but are likely to be outstripped by the overall growth in the market. Similarly, the “greening by IT” developments discussed here will have an effect on overall resource use, but it is likely that the best that can be achieved is to slow the rate of increase of demand, not to achieve an absolute reduction.

Whether this will continue into the future remains to be seen. The pessimistic view is that we will continue in a business-as-usual fashion and will be unable to change our behavior until it is too late. A more optimistic alternative to this scenario relies on our capacity for innovation to stop the problem. The true optimist hopes that a combination of our ingenuity and our realization of the risks will bring about changes before the consequences become unbearable (although noting that this point has been reached for some parts of the world, affecting the plant and animal species that live there).

References

1E, n.d 1E, n.d. The PC energy report. http://www.1e.com/energycampaign/downloads/1E_reportFINAL.pdf.

Alcott B. Jevons paradox. *Ecol. Econ.* 2005;54(1):9–21.

BBC. *How Dutch Team is 3D-Printing a Full-Sized House.* http://www.bbc.co.uk/news/technology-27221199. 2014 (03.05.14).

Bilich A. *Improving the Precision and Accuracy of Geodetic GPS: Applications to Multipath and Seismology.* (Ph.D. Thesis) University of Colorado; 2006.

Blodget H. *The Number of Smartphones in Use is About to Pass the Number of PCs.* http://www.businessinsider.com/number-of-smartphones-tablets-pcs-2013-12#ixzz3CdoVKxAX. 2013 (11.12.13).

Chiu M, Loffer M, Roberts R. *The Internet of Things.* http://www.mckinsey.com/insights/high_tech_telecoms_internet/the_internet_of_things. 2010 (March 2010).

Cisco, 2014. http://www.cisco.com/c/en/us/solutions/collateral/service-provider/ip-ngn-ip-next-generation-network/white_paper_c11-481360.pdf.

Coroama V, Hilty L, Heiri E, Horn F. The direct energy demand of internet data flows. *J. Ind. Ecol.* 2013;17(5):680–688.

Covert A. *Microsoft is About to Take Windows XP Off Life Support.* http://money.cnn.com/2014/01/29/technology/enterprise/windows-xp/index.html. 2014 (January 2014).

Doyle A. *The Great Boer War.* London: Smith, Elder & Co; 1902.

Gartner. *Gartner Highlights Key Predictions for IT Organizations and Users in 2010 and Beyond.* http://www.gartner.com/newsroom/id/1278413. 2013 (13.01.10).

Greenpeace. *How Dirty is Your Data?* http://www.greenpeace.org/international/Global/international/publications/climate/2011/Cool%20IT/dirty-data-report-greenpeace.pdf. 2013 (April 2011).

NASA. *Building a Lunar Base with 3D Printing.* http://sservi.nasa.gov/articles/building-a-lunar-base-with-3d-printing/2014. 2014.

Raghavan B, Ma J. *The Energy and Emergy of the Internet.* http://www1.icsi.berkeley.edu/~barath/papers/emergy-hotnets11.pdf. 2014.

Rose W, Rouse T. *Sub-National Electricity Consumption Statistics and Household Energy Distribution Analysis for 2010.* https://www.gov.uk/government/uploads/system/uploads/attachment_data/file/65933/4782-subnat-electricity-cons-stats-article.pdf. 2012 (December 2012).

Stonington J, Wong V. *The 20 Countries with the Highest Per Capita Cell-Phone Use, Business Week.* http://images.businessweek.com/slideshows/20110213/the-20-countries-with-the-highest-per-capita-cell-phone-use. 2011 (14.02.11).

Venkataraman A. *Global Census Shows Datacentre Power Demand Grew 63% in 2012.* http://www.computerweekly.com/news/2240164589/Datacentre-power-demand-grew-63-in-2012-Global-datacentre-census. 2013 (08.10.12).

Wendt J. *A Zettabyte of Data Puts New Premium on Scale-out Storage Solutions.* http://www.dcig.com/2011/07/a-zettabyte-of-data-puts-new-premium-on-scaleout.html. 2011 (20.06.11).

World Bank. *Digital Finance: Empowering the Poor via New Technologies.* http://www.worldbank.org/en/news/feature/2014/04/10/digital-finance-empowering-poor-new-technologies. 2014 (10.04.14).

CHAPTER 3

Measurements and Sustainability

Eric Rondeau; Francis Lepage; Jean-Philippe Georges; Gérard Morel Université de Lorraine, Vandoeuvre-lès-Nancy, France

Abstract

The design of ICT-based application traditionally focuses on the analysis of relationships between ICT performances, user or application requirements and costs. However, the climate change, the environmental pollution, the earth resource depletion, the energy price increase forced ICT sector to analyse and to consider these issues as new requirements in the development of ICT solutions. The specification of environmental measurements is then necessary in order to be able to correctly assess the performances of so-called green ICT systems. In this chapter, ICT metrics and environmental metrics are presented. The social responsibilities of ICT companies in the context of sustainable development are then developed. Finally, from an example, a general methodology for eco-designing ICT systems is proposed. It is based on systems engineering enabling to analyse the system as a whole in integrating the metrics previously described.

Keywords

Green ICT

Green KPI

ICT Eco-design

Computer ethic

Systems engineering

Introduction

Information and communications technology (ICT) engineers are developing and creating a virtual world offering new services and new applications to help people both in their work and their daily lives. Nevertheless, ICT engineers are almost always constrained in their project development by the intrinsic hardware performances of calculators, storage systems, and communication systems. The monitoring and measurement of physical ICT system performances are crucial to assess the computer processing unit (CPU) load, the available memory, the used bandwidth, and so on to guarantee the ICT-based services correctly work regarding their expected use. The famous Moore’s law states that the number of transistors on a chip tends to double every 18 months; this enabled a rapid growth of digital system performances. However, these technical performances do not provide direct information about the level of quality of the offered services. In the International Telecommunication Union-Telecommunication (ITU-T) standardization sector’s E.800 recommendation (ITU-E800, 1994), quality of service (QoS) is defined as “the collective effect of service performances, which determine the degree of satisfaction of a user of the service.”

The satisfaction of users is then the key of ICT business and must be specified in a service-level agreement (SLA) signed by ICT experts and their customers. By definition, SLA is not a technical document and must be understandable by all stakeholders who are not necessarily aware of ICT terms. Two major issues should be analyzed during the SLA specification: the identification of relationships between the performance indicators defined by the user’s application and ICT technical performances and one related to the monitoring of these indicators to be sure that the contract is fulfilled.

The purpose of translation between ICT and the user’s application performances covers many types of problems. The translation can be direct such as having the application response time correspond to the time for ICT experts to process and transport the applicative request. In this case, the main problem is to clearly define the context of this requirement in terms of number of users, opening hours, and so on and to characterize the type of delay (worst case, average, confidence interval, etc.). However, the translation can be more complex when the user’s requirements are expressed in specific professional terms. For example, the control of an industrial process is assessed by analyzing the stability of its behavior around a set point to be reached. In the research on networked control systems, the identification of impact of ICT performances (and especially network) on industrial process stability requires complex preliminary studies and development of new approaches (Vatanski et al., 2009). One other barrier in the SLA definition is the specification of the user’s requirements with qualitative, not quantitative information. The user’s perception of the quality of a phone call, TV broadcast, Web site, and so on is subjective and complicated to analyze and to associate with quantitative ICT parameters (delay, jitter, etc.). Usually, the user’s perception is transformed in a metric based on mean opinion score using a scale between 0 (no service) and 5 (perfect service) to guide the ICT experts in their technical configurations.

Monitoring indicators specified in SLA is essential to identify the border between the ICT systems and the application itself. The goal is to be able to understand and identify the cause of malfunctions and to determine the stakeholders’ responsibilities. ICT systems must be continually supervised to analyze their performances in order to detect, anticipate, and recover faults. The assessment of ICT performances can be based on measures or models or a combination of both.

Basically, the measurement requires probes, a monitoring system, and standardized protocols to access the whole metrics. Usually, a management information base (MIB) is implemented for each piece of equipment (computer, printer, switch, router, and so on) and supports in a standardized hierarchical structure all the equipment properties (name, OS, version, storage capacities, bandwidth, etc.). The monitoring systems (such as Nagios, Centreon, etc.) can then collect or modify the information stored in MIB by using simple network management protocol (SNMP). This approach is apparently easy to implement, but in practice the selection of pertinent information (regarding the SLA contract) defined in MIB is a complex process. Instead of measuring the parameters of equipment, another solution is to directly analyze the performances of the application or service defined in SLA. For that, robots are developed and simulate the user’s behavior using the application. In all cases, one inherent issue of measurement is its intrusiveness with two consequences: (1) each request for a monitoring ICT system consumes CPU and bandwidth and has an impact on its performance and (2) the response time of a monitoring request depends on the performance of the ICT infrastructure.

Another approach to assess ICT performances is to use mathematical theories from one of two methods, constructive and black box. The constructive methods are based on the assembly of elementary components with specific properties; their combination can be used to estimate average delays, average buffer occupation (queuing theory) or bounded delays, and backlog bounds (network calculus theory) (Georges et al., 2005). In the second method, the ICT system or a part of it is considered as a black box and its behavior (output) is analyzed regarding the changes of ICT parameters (input). From this analysis (i.e., experiment design method), a model of the ICT system can be defined. The black box method is less generic than the constructive method, but it is better correlated with real ICT properties.

With a monitoring system, it is very interesting to couple the measures and models. The models represent the expected behavior of an ICT system and the measures its real behavior. A difference between models and measures can be used to detect anomalies and to anticipate faults according to a trend analysis. This combination of models and measures is one way to develop a successful monitoring system.

Since the birth of computer science and telecommunications, their performance evaluation mainly focused on technical and cost indicators. But, as explained in the beginning of this introduction, the final goal of ICT is to facilitate people’s lives without undesirably affecting either their health or quality of life. Therefore, these effects also should be assessed during the full life cycle of an ICT product or ICT-based solutions in considering its manufacturing step, its use step, and its next life. An overall assessment of pollution must be performed to determine the ICT carbon footprint, the toxic material rate used in ICT devices, and so on, which impacts the health of people. Moreover, the Earth’s resources used for ICT must be continually decreased to preserve the quality of life of current and future generations. The preservation of the Earth’s resources includes recycling and extensive use of renewable energy. On another issue, quality of life is also related to ethical questions for both employees in ICT companies and ICT users with general considerations such as the salary of employees, the gender balance, and so on, or questions more specific to the ICT area such as the protection of privacy and personnel data.

In summary, ICT must be assessed using the three Ps or pillars of sustainable development (Figure 3.1) during the engineering process of the target system as a whole by balancing people, planet, and profit requirements with the final objective to design green ICT solutions.

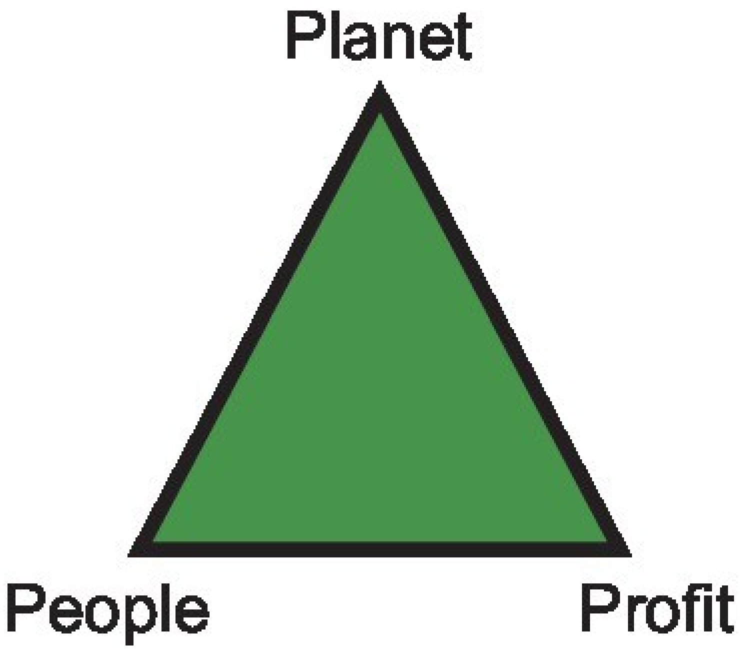


FIGURE 3.1 The three pillars of sustainable development.

ICT engineering should be systemic in order to analyze the negative or positive effects among the three pillars. For example, the mitigation of data center energy consumption is interesting both in terms of the environment and profit. But increasing data center capacity to grow business activities consumes more energy, generating a negative impact on the planet. The well-known analysis using a C2C fractal tile tool (Donough and Braungart, 2002) is a new challenge and obligates ICT engineers to study the solution spaces not only on business and technical performances but also in associating new metrics coming from ecology and ethics. Thus, the SLA specifications are more complex in integrating additional requirements. This complexity can be managed in using systems engineering to guarantee that the development of ICT products, services, and ICT-based solutions is sustainable and measurable for validation and verification of all technical solutions. The sustainability is not only expressed in term of longevity of solutions but also must include properties of modularity, flexibility, scalability, and recyclability.

The objective of this chapter is to show the different aspects of ICT metrics. The chapter is then organized as follows: Section "ICT Technical Measures" briefly explains the traditional metrics used to assess intrinsic ICT performances. Section "Ecological Measures And Ethical

Consideration” focuses on the performance indicators specific to the environment and ethics. From a simple ICT architecture, Section “Systems Engineering for Designing Sustainable ICT-Based Architectures” presents systems engineering approach to specify and to consider the measures in Green ICT-based solutions. Section “Conclusion” concludes the chapter.

ICT Technical Measures

Introduction

The measurement of ICT systems can be compared with the measurement applied to any natural or artificial system of the world. However, the main difference with other systems is that ICT measures are related to data performance, which is an abstract thing. In general way, ICT functional model can be described as in Figure 3.2a in considering an ICT system as a black box. The input and the output are data crossing the ICT system. Controllers can be implemented over the system to manage the ICT QoS such as scheduler, smother, load balancer, and so on in order to meet the user’s requirement. Finally, energy is the resource required for supplying an ICT system. An ICT system offers three elementary types of services: data processing, data transport, and data storage. Moreover, a set of ICT systems can provide global services such as multimedia. Traditionally, ICT system performances are assessed by observing the deviation between the requests of service sent by the user containing the input data and the corresponding response of service supporting the output data. This deviation characterizes the service offered by ICT system to the user. Finally, until very recently, energy has not been analyzed and is considered an infinite resource with no impact on cost and environment.

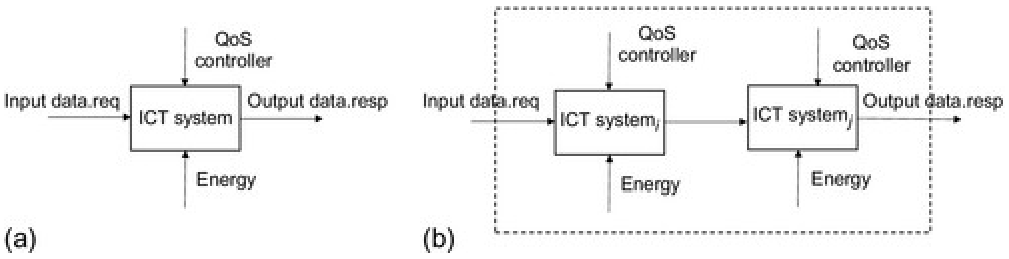


FIGURE 3.2 ICT system functional models.

The main attributes used in the data measurement are quantity, throughput, quality, availability, and security. However, in general, the measure of these attributes is not directly useful in assessing ICT systems. *Metric* is a concept defined as a measure of an attribute or a combination of attribute measures that provides pertinent information. One example of an ICT metric is the available communication bandwidth. It is expressed in bits per second, and its value is obtained by using models and several measurements.

Moreover, the assessment of a complex system composed of subsystems (Figure 3.2b) requires derived metrics that can be developed from a combination of basic metrics relative to each subsystem. The three classes of operation to obtain derived metrics are additive, multiplicative, and concave. The additive metric is calculated by adding the basic metric of each part of the system. The one-way delay of a path calculated by summing the one-way delay on each link of the path illustrates the principle of additive metric. The multiplicative metric is calculated by multiplying metrics of each part of the system. The metric of packet loss ratio can use the multiplicative operation to assess a complex architecture. The concave metric is determined by selecting the minimum value of each part of the system. For example, the estimation of end-to-end bandwidth is constrained by the link with the lowest bandwidth.

A common metric applicable to all ICT systems is *response time* corresponding to the delay between service request arrival time in the system (input) and service response available (output). It depends on both type of service and the data amount in the request. The response time is sometimes called *delay* or *latency*. The delay is calculated from two measures obtained from different devices and then requires synchronizing their clocks or using the same clock. In the next sections, the different types of services are described.

Service of Data Processing

Data processing is characterized by two specific metrics: the processing capacity and the processing quality.

The *processing capacity* is the higher number of requests per time unit that could be sent (without loss of request) to ICT system input in order to be processed. Service tests (mathematically complex functions) are defined to compare the processing capacity of different services.

The *processing quality* is determined by the accuracy of the results. This metric is very important in scientific computation.

Service of Data Transport

The data transport service uses three specific metrics: the bandwidth, the jitter, and the packet loss rate (Michaut and Lepage, 2005). There are two types of bandwidth. The first type is the *total path bandwidth*, also called *path capacity*. It expresses the maximum total throughput accepted by the path. The second type is the *available bandwidth* expressing the maximum throughput offered to the user. Thus, the available bandwidth is the path capacity minus the used traffic (concurrent traffic). The bandwidth measure is based on interval packet times.

Jitter is the delay variation between the same successive requests. It is not a measured value but is obtained by subtracting the delays of two successive requests.

The reliability of a communication network path is expressed by the *packet loss rate*. This metric is equal to the number of packets not received divided by the total number of packets sent. It should be noted that erroneous packets are not generally considered outside the lost packets in computer networks because most applications require data integrity. Indeed, the frame reception driver discards each erroneous packet that becomes lost. However, this is not a general rule, especially in audio or video networks, because applications can accept a low error rate. The measurement is achieved by using a sequence number in each packet and counting the missing numbers.

Service of Data Storage

Data storage includes two metrics: the storage capacity and the data throughput. The *storage capacity* is difficult to measure using the service black box approach. Fortunately, specific requests provide the total or remaining capacity of the storage system.

Data throughput is sometimes different for read and write operations. For a write operation, the measure is achieved by counting the maximum amount of data that the service accepts per time unit. For a read operation, the measure is similar but is applied to the service output. The time unit must be large enough to integrate gaps between storage units.

Multimedia Service

A multimedia service must allow the user to play multimedia data from voice or video acquisition crossing a chain of ICT systems. Two methods are used, the full reference and the no reference.

The *full reference (FR)* method consists in comparing the quality of reference signal from the source (perfect signal) with the received signal (degraded signal). FR measures deliver the highest accuracy and repeatability. However, FR measures can be applied to dedicated tests only in live networks.

The *no reference (NR)* method uses a degraded signal without information about the reference signal. From the NR method, some characteristics of a conversation, such as voice, gender, background noise, and so on, can be identified.

Conclusion

The fundamental activity of an ICT engineer is to configure ICT systems to offer its customers the best or optimized services regarding the main ICT metrics described in this section. However, the recent high increase in energy prices and the new environmental awareness by politicians and people require an ICT engineer to consider energy in SLA, which has been renamed *green SLA*. For example, Laszewski and Wang (2010) and Makela and Luukkainen (2013) present new metrics on energy, especially in the context of green SLA for data center. Moreover, power over ethernet (PoE) technology is a new network service enabling the transport of energy on data link to remotely supply ICT products and finally to control the status of ICT devices by using different sleeping modes. Thus, energy becomes an integral part of ICT metrics. However, limiting green SLA to only an energy metric is very restrictive. The goal of the next section is to describe all the facets of green ICT.

Ecological Measures and Ethical Consideration

Introduction

The consideration of the three pillars of sustainable development in the design of ICT-based solutions is complex because many new parameters should be added during the design process and because the limit of study space is always difficult to determine. Indeed, an enterprise involved in a sustainable development approach must analyze the environmental impact of its design process itself (project system) and of the complete life cycle of designed systems (system of interest). The issue is to obtain a global result in integrating and estimating not only the enterprise's environmental footprint but also in including all stakeholders (subcontractors, suppliers, sellers, users, recycler, etc.) involved in the enterprise activity. The goal is to prevent the enterprise from creating a marketing communication about its green and ethical virtues in keeping its environmentally friendly activities and in outsourcing the others. In this context, standardization is a major aspect to successfully reach this objective to ensure seamless environmental monitoring between the different stakeholders. The International Organization for Standardization (ISO) 14001 standard provides a framework for an enterprise to follow to set up an effective environmental management system. However, the purpose of ISO 14001 is not to estimate environmental performances and cannot explicitly show whether the enterprise pollutes.

The impact of an enterprise on the environment should be determined by using a list of key performance indicators (KPIs). A KPI is a metric or measure used to quantify and evaluate an organization's performance in relation to meeting of targets and objectives. Many standardization organizations or working groups have proposed environmental KPIs. The ITU report "General Specifications and KPIs" (ITU-T report, 2012) referred to an impressive list of initiatives specifying KPI. There are global initiatives such as the Carbon Disclosure Project, Dow Jones Sustainability

Index, GHG Protocol corporate standard, Global Reporting Initiative, ISO 14031, ISO 14064-1 and initiatives from ICT sectors such as European Telecommunications Network Operators Association, European Telecommunication Standards Institute, the Green Grid, GSM Association, International Electrotechnical Commission (IEC), International Telecommunication Union—Study Group 5 (ITU-T SG5). In conclusion, the 2012 ITU-T report does not propose a KPI harmonization but suggests a general process to define environmental KPIs. This process has three steps: defining the needs, listing relevant KPIs, and verifying third-party data collected for all environmental indicators. This method fully respects the basis of systems engineering developed in Section "Systems Engineering for Designing Sustainable ICT-Based Architectures" of this chapter.

The standardization requirement concerns not only the definition of environmental KPI but also the specification of protocols enabling the collection of KPI and control of ICT-based applications regarding the KPI status. The IEEE P1888, "Ubiquitous Green Community Control Network Working Protocol" (2011) is an example illustrating the standardization of remote control architecture for digital communities, intelligent building groups, and digital metropolitan networks in terms of the energy, environment, and security domains.

The intention of this section is not to describe the list of environmental KPIs defined in the different working groups (mentioned previously) but to classify KPIs according to three categories—pollution, Earth's resources, and ethics—to present all facets of sustainable development.

ICT Impact on Pollution

By definition (Chapman, 2007), c*ontamination* is simply the presence of a substance where it should not be or concentrations above specified levels. That is, pollution is a contamination that results in or can result in adverse biological effects to communities. It means that all pollutants are contaminants, but not all contaminants are pollutants. Various forms of pollution can affect the air, soil, light, noise, heat, vision, and water. All these forms of pollution can be generated by the ICT sector. Therefore, it is essential to identify pollution causes to mitigate or to eliminate their impact on environment.

ICT products are composed of many chemical substances that are toxic for the environment and people. For example, beryllium used in relays is dangerous for workers manufacturing this electronic equipment, bromated flame retardant used in mobile phones is neurotoxic, cadmium used in the rechargeable computer batteries is toxic for kidneys and bones, mercury in the flat screens can affect the brain and central nervous system, and so on. The management of these chemical substances during the life cycle of electronic products is essential in the green context and must include informing manufacturers, users, and recyclers about the presence of toxic elements. Especially, in the end-of-life product cycle, electronic wastes including hazardous products pollute soil, atmosphere, water, and the beauty of landscape. Incineration releases heavy metals and ashes into the air. Obviously, there are two ways to limit this pollution. One consists of mitigating the use of hazardous elements in manufacturing electronic equipment, and the other pertains to the proper management of dismantling and recycling steps. The GS1 EPC global standard in the consumer electronics supply chain (GS1, 2010) is an interesting solution to ensure the traceability of electronic equipment by using radio frequency identification (RFID) technologies, which provides storage for a cartography of an electronic product's components. A through knowledge of electronic product composition is essential for improving the efficiency of hazardous materials management and for preventing health issues of employers working on recycling processes and have direct contact with electronic wastes. Kubler et al. (2014) propose a similar concept to embed RFID in material for improving the monitoring of the life cycle of materials.

"If the cloud were a country, it would have the fifth largest electricity demand in the world" (Green Peace International Report, 2012). The Smart 2020 report (Smart 2020, 2008) provided quantitative information on carbon emission produced by ICT by considering the complete electronic product life cycle including the ICT use step and the manufacturing and dismantling steps (embodied carbon). The energy consumed by the ICT sector significantly adds carbon emissions to the atmosphere, causing air and thermal pollution.

The energy consumed by ICT use corresponds to the power supply to feed ICT equipment (computers, networks, etc.), cooling systems, and energy transport. The installation of a base transceiver station (BTS) in the desert is a typical example that illustrates clearly the logistical chain to be considered in the energy consumption estimation. It includes fuel to produce electricity from generators. This electricity is used both for BTS and cooling systems. Finally, it is also necessary to manage a crew of tank trucks consuming fuel to transport it to BTS generators. Another interesting example is the study of data center energy consumption. Emerson Network Power (2009) defines two categories of energy use: the demand side and the supply side. The demand-side systems are the servers and storage, communication, and other ICT systems that support a business. The supply side systems include uninterruptible power supplies (UPS), power distribution, cooling, lighting, and building switchgear. The power usage effectiveness (PUE) proposed by Green Grid (2012) to assess data center efficiency is the ratio of facilities energy (supply side) to IT equipment energy (demand side) and is expressed as follows:

$$\mathrm{PUE} = \frac{\text{Total Facility Energy}}{\text{IT Equipment Energy}}$$

However, as Ascierto (2011) mentioned, PUE is a simple metric that does not reflect the entire complexity of data centers regarding its global efficiency. Especially, PUE does not provide performance on its use (bandwidth, bytes, etc.).

Nevertheless, the interest in identifying all energy consumption sources is not only to control the global energy consumption of data center but also to develop a strategy to mitigate the energy consumption. Indeed, Emerson Network Power (2009) shows the cascade of effects between component chain in explaining that the energy reduction of a server has an impact on the power supply, which has an impact on power distribution, which has an impact on UPS and on cooling, and so on. Emerson Network Power estimates 1 W saved at the processor saves approximately 2.84 W of total consumption.

A thorough knowledge of these interactions is crucial in decreasing energy consumption. In this process, two criteria concerning energy and carbon emission must be clearly identified. The mitigation of energy in ICT is mainly a business objective corresponding to the profit pillar of sustainable development. As mentioned in (Emerson Network Power, 2009), the global electricity prices increased 56% between 2002 and 2006 and continue to growth. Thus, ICT sectors must lessen their energy consumption to limit the impact of energy costs on their business.

Carbon emission participates in the increasing of greenhouse gas (e.g., as methane) and is related to the planet pillar of sustainable development. Even if carbon emission is correlated to the energy consumption, its decrease regarding the planet criterion and the profit criterion can provide two opposite strategies. Indeed, the selection of a clean energy to reduce pollution can increase the electricity prices. Moreover, the decrease of carbon emission is complex, and its complexity will increase with smart grid development because of the variation of carbon rate in the energy production. The carbon emission algorithms should integrate these variations. The French Company of Energy Transport (RTE) provides such information in real time that can be used to develop optimal strategies in the usage management of ICT equipment in an enterprise. For example, the observation of a peak on carbon to produce electricity could automatically configure the laptops in battery mode.

The Smart 2020 report also analyzes the embodied carbon in carbon emissions during both the manufacturing of ICT components and their end-of-life treatment. In general, RICS QS & Construction Standards (2012) explain that the extraction, manufacture, transportation, assembly, replacement, and deconstruction of construction materials or products involve embodied carbon.

The knowledge of embodied carbon and the carbon emitted during the usage phase enables the determination of the sustainability of an ICT-based solution or product and then an estimation of its optimum obsolescence. From this information, the global energy consumption of an active ICT product can be compared with a new generation of ICT products using more energy-efficient components to anticipate the ICT installation renewal. The approach in extending the ICT product life at all costs may be in opposition to the sustainability objective of promoting environmental-friendly solutions. The concept of point or band of optimum obsolescence was explained by Tuppen (2013) to define the right time to change a car, a notebook, and so on.

Electrosmog is the electromagnetic radiation emitted by electronic equipment (computers, mobile phones, etc.) and is a hot topic in the scientific community to determine its real effect on health and then to determine whether electrosmog is a form of pollution or a contamination. The health effects of radio waves were intensively studied by national and international organizations including the International Commission on Non-Ionizing Radiation Protection, the Institute of Electrical and Electronics Engineers (IEEE), and the World Health Organization (WHO). The effects of electromagnetic fields on the human body depend not only on their field level but also on their frequency and energy. All study results indicated that the unique noncontroversial effect of nonionizing electromagneting field (EMF) is thermal (WHO, 2006, 2011). To avoid it, all organizations have determined the maximum acceptable values for EMF (IEEE, 2005) associated with the frequency range of the radio frequency (RF) channel. Based on these recommendations, the government of each country or state has defined the legal maximum level of EMF generated by wireless network antennas and their maximum specific absorption rate (SAR) value. SAR is a measure of the maximum energy absorbed per unit of mass of the head of a person using a mobile phone. These values are sometime lower than the maximum. It is not disputed that electromagnetic fields above certain levels can trigger biological effects. For example, weak electromagnetic transmitters with a frequency spectrum between 0:1-10 MHz can affect animal behavior, particularly the orientation of migrating birds (Ritz et al., 2004). But biological effects do not necessarily cause health hazards. However, researches are actively continuing to confirm that low level, long-term exposure to radio frequency fields do not generate adverse health effects.

We now explain many proposals to limit electrosmog. One way is to use cognitive radio-enabling automatic detection of the available channels in a wireless spectrum and accordingly to change their transmission and reception parameters (Stevenson et al., 2009). Another way is the smart antenna technique, which uses spatial multiplexing and coding for removing interferences (Cui et al., 2005). Other solutions consist of implementing base stations and mobile phones using low power transmissions. However, this approach tends to multiply (for an equivalent level of QoS) the number of base stations and then to increase the energy consumption and CO_2 pollution while spoiling the esthetic environment. Based on this idea, Cerri and De Leo (2004) propose to reduce electromagnetic pollution of mobile communication systems in optimizing radio base station locations. Finally, a good improvement in wireless communication is to put antennas in fake chimneys or fake trees. This is a good way to avoid the electromagnetic hypersensitivity effect (Seitz et al., 2005) and to minimize the visual pollution.

Two additional aspects related to pollution induced by ICT, but not really considered in green ICT, are light and noise pollution. A recommendation has been made to limit paper documents to preserve the environment. The systematic use of electronic documents is often recommended. However, computer screens generate a light pollution affecting health. The computer vision syndrome defined by the American Optometric Association includes sore, dry, irritated, or watery eyes; headaches; sleep disorders; and so on of people using computers. A specific information on that could be mentioned in each proposed ICT solution.

The noise pollution generated by cell phone ringtones and conversations in public area has also impacted quality of life. This problem is related to ethical behavior in the use of ICT and is more complex to measure and assess.

Resource Efficiency

Janine Benyus explains in the last chapter of her book *Biomimicry* (Benyus, 2002) “how will we conduct business,” in an ecosystem that is a complex of living organisms, their physical environment, and all their interrelationships within it in a particular unit of space (Encyclopedia Britannica, 2012). The evolution of ecosystems generally occurs in two phases: the developing stage and the mature stage (Allenby and Cooper, 1994). The developing stage involves few species and short food chains. At this stage, the ecosystem is unstable but highly productive, in the sense that it builds organic matter faster than it breaks it down. The mature ecosystem is more complex, more diversified, and more stable. Currently, the business model used in our society, especially in the ICT sector, is the developing stage. The challenge is to move to the mature stage.

Businesses based on the developing stage consider natural resources to be infinite. However, different studies have estimated that the earth reserves of a number of materials (Diederen, 2009; Cohen, 2007; Kesler, 2007). For example, the reserve of Indium is between 5 and 10 years, of copper 20-50 years, and of gallium 5 years. Indium, copper, and gallium are components used in semiconductors and there are two actions to be able to manufacture ICT products in long term. The first one consists of substituting other materials having the same properties; the second one is to generalize the recycling of ICT products. “A tonne of gold ore yields 5 g of gold, compared to a staggering 400 g yielded from a tonne of used mobile phones” (ITU-T report, 2012). In this context, the interest in recycling is both ecological and economical. The EuPs report (2007) provides much interesting information about materials inside computers and screens.

Recycling in the ICT sector is a big issue because of the rapid obsolescence of hardware and software, which continually offer new functionality with better performance, lead to premature equipment renewal, and produce electronic waste. The different recycling aspects include the reuse of old equipment in other applications and in equipment dismantling. For example, reuse might involve replacing an old mobile phone with a new one and identifying a market for selling the old one to people with different needs. In the book *Cradle to Cradle*, Donough and Braungart (2002) propose that products be designed with their raw materials separated into biological nutrients and technical nutrients. The interest is in avoiding the design of “monstrous hybrids” and in facilitating the recycling step by having one part dedicated to biological metabolisms and another part dedicated to technical metabolisms (enabling the recovery of rare materials, for example). This aspect is very important in the context of Internet of things when electronics and batteries are more and more mixed with the natural environment and its dismantlement should be carefully analyzed during its design.

Standardization and legislation have a primary role in the success of recycling. For example, the Waste Electrical and Electronic Equipment Directive (WEEE, 2003) in the European Community (2002/96/EC) imposes responsibility for the disposal of electrical waste and electronic equipment on the “manufacturers.” Other examples for limiting e-waste are the recent ITU recommendations (ITU, 2011; ITU-T L.1001, 2012), proposed to specify universal power adapter and charger solutions for both mobile and network devices enabling consumers to reuse them when they buy new mobile phones or other electronic equipment.

The limitation of natural reserves (Kesler, 2007) also concerns energy (oil, gas, etc.) and suggests the use of renewable energies (solar, wind, water, etc.) to preserve the earth’s resources. Moreover, this method can be also applied to energy and materials. Electronic equipment consumes energy and dissipates heat. This heat can be reused as a resource for other applications. For example, the heat emitted by a data center can provide heat for buildings (Val-Europe, 2012).

“Water is life, sustaining ecosystems and regulating our climate. But it’s a finite resource, and less than 1% of the world’s fresh water is accessible for direct human use” (European Commission, 2010). The water scarcity defined as the lack of sufficient available water resources to meet the demands of water use within a region is another large challenge to the green ICT. As Lewis (2013) explained, the ICT sector has a strong demand for energy; data centers and the energy sector consume large amounts of water. Moreover, the air cooling in data centers has recently replaced by a water cooling system because water conducts more heat than air, and warm water can be reused more easily to heat buildings and swimming pools near data centers. For measuring the water consumption in data centers, the Green Grid uses the metric water usage effectiveness (WUE) corresponding to the annual water use divided by IT equipment energy; it is expressed in liters/kilowatt-hour. This metric is applied to the site consumption WUE_{site} and to the source consumption $\text{WUE}_{\text{source}}$. Nevertheless, the manufacturing, transport, and recycling of water consumed during the life cycle of ICT products should be assessed.

Main Green Measures of Performances

From the previous analysis, the objective now is to define three main metrics to measure of performance (MoP) in the context of system engineering, enabling the development of green ICT regarding recyclability, energy consumption, and carbon emission. Obviously, these MoP are first proposals (Drouant et al., 2014) and should be refined and completed by others in order to take into account of all kinds of pollution and resource scarcity issues. Table 3.1 provides all notations used in the different equations.

Table 3.1

Notations

Symbol	Description
X	Set of equipment for the architecture
$\|X\|$	Number of items of equipment in the architecture
X_s	Set of substitute equipment
X_r	Set of repackaged equipment: $X_r \in (X \cup X_s)$

Symbol	Description
$\overline{X}_r$	Set of used equipment never repackaged: $\overline{X}_r = (X \cup X_s) - X_r$
ρ_i	Recyclability rate for an item of equipment *i*
Γ	Recyclability rate of the whole architecture
E	Energy consumed by the whole architecture
E_m	Energy consumed by the equipment’s manufacturing process
E_u	Energy consumed by the whole architecture during its use phase
P_u	Power consumption of the whole architecture during its use phase
E_d	Energy consumed during the dismantling phase of $\overline{X}_r$
f	Factor related to the environment (air, feed efficiency)
h	Number of operating hours per year
δ	Power consumption gain (relative to the traffic profile)
ω_i	Traffic load for a switch port *i*
ε_i	Power consumption of an item of equipment *i*
$\varnothing$	Power consumption of a switch in the idle state (no traffic)

Symbol	Description
σ	Power consumption of a busy switch port (full load)
Φ	CO_2 emission for the whole architecture
α	Multiplicative gain between energy consumed and CO_2 emission by country
τ	Multiplicative gain caused by energy transportation losses

MoP on Recyclability

This metric in Equation (3.1) involves the set X_r of equipment that will never be repackaged or refurbished for use in another architecture. In the worst case, it corresponds to $|X|$ the total number of items of ICT equipment used in the ICT-based architecture plus X_s equipment used to substitute for existing ICT equipment. Items of ICT equipment might be replaced because of failure, upgrades, or even extensions to the architecture. It is important to note that the list of equipment in Table 3.1 must be considered in terms of the entire architecture life cycle.

The recyclability rate of an item of equipment (ρ_i) ranges from 0 to 1 (1 corresponds to 100% recyclability and means that the item is fully reusable as a resource in other applications). From Equation (3.1), the management of metal can be taken into account in ρ in using the recyclability rate of metals defined in Graedel et al. (2011), for example, and is referred to as the *end-of-life recycling rate* (EOF-RR), including recycling as a pure metal (e.g., copper) and as an alloy (e.g., brass).

$$\Gamma = \frac{\sum_{i \in \overline{X}_r} \rho_i}{|\overline{X}_r|}, \quad \overline{X}_r = (X \cup X_s) - X_r \tag{3.1}$$

To optimize this metric, it is necessary to select highly recyclable ICT equipment, to anticipate any repackaging, and to limit the number of items of ICT equipment to be implemented except for those that are 100% recyclable ($\Gamma = 1$). Even if it does not mean zero energy consumption corresponding to the MoP on energy consumption, 100% recyclability is an ideal objective for a circular economy (Donough and Braungart, 2002).

MoP on Energy Consumption

In the estimation of energy consumption in Equation (3.2), there are three stages, namely the manufacturing of the ICT equipment (E_m), the use of the ICT-based architecture (E_u), and the dismantling of ICT-based architecture (E_d). It is important to note that the objective of green ICT is to reduce the energy consumption. With Equation (3.2), the dismantling step has a negative impact on this MoP to limit the effort of recycling. For this, Williams and Sasaki (2003) propose a similar equation with a negative term for (E_d) to indicate the positive gain of recycling. However, Equation (3.2) should be globally analyzed. A company developing an efficient management of its electronic waste will actually consume energy for the dismantling but will mitigate the energy consumption for manufacturing its new ICT products in avoiding the extraction of new earth resources. Thus, the optimization is not based on one product life cycle but on the manufacturing of multiple products always using the same earth resources. Another interpretation of Equation (3.2) is to recommend the reuse of the obsolete products in other applications or for other users before recycling.

$$E = E_{\mathrm{m}} + E_{\mathrm{u}} + E_{\mathrm{d}} = E_{\mathrm{m}} + \int_{t=0}^{\text{end of life cyle}} P_{\mathrm{u}}(t)\mathrm{d}t + E_{\mathrm{d}} \tag{3.2}$$

The energy used by the transport of ICT products from the manufacturer to the customer is included in (E_m). In the same way, the energy consumed to transport an ICT product for recycling is associated with (E_d).

P_u represents the power consumption by the ICT-based architecture during its use phase. Many research works propose expressions for modeling the energy consumed by ICT equipment. There are two main classes of proposal. One involves high-level modeling in which the interest is in approximating the energy consumption of general network architectures. The other is specific to network technologies and provides precise outcomes.

Two examples in the network domain illustrate these two classes.

Firstly, Foll (2008) considers a high-level model without specifying a networking technology. The objective is to develop a management tool for the Orange Company that offers higher visibility for the current and future consumption of its network. In this context, the following macroscopic model is defined per year:

$$\int_{t}^{\text{one year}} P_{\mathrm{u}}(t)\mathrm{d}t = \sum_{i \epsilon X} \varepsilon_i \times f \times h \tag{3.3}$$

Secondly, Reviriego et al. (2012) propose a more specific equation for estimating the energy consumption of a network architecture based on switched-ethernet technology. The model for the energy-efficient ethernet switch is:

$$P_{\mathrm{u}} = \varnothing + \sigma \sum_{\mathrm{Port}\, i} \min(1, \delta\omega_i) \tag{3.4}$$

In Equation (3.4), σ represents the difference between the power consumption at full load and the traffic-free power consumption divided by the number of ports in the ethernet switch.

P_{u} can be also approximated by a simple measure of ICT product or by using information provided by the Energy Star label regarding the efficient energy consumption of computers, displays, and imaging equipment. However, the understanding of the relationship between both the network activities and configuration and its energy consumption is open and is very important regarding two issues. Firstly, a formal relationship will enable proposing ICT-optimized strategies to mitigate the energy consumption. Secondly, a formal relationship could be compared with the current monitoring of ICT infrastructure in order to develop smart sensors able to observe deviations with expected results and to automatically detect possible anomalies.

MoP on Carbon Emission

Carbon emission is estimated from the energy used during the entire life cycle of an ICT product or ICT-based architecture and from the information in Table 3.2 matching rate between energy consumption and CO_2 emission in electricity production for some European countries.

Table 3.2

Factors for kg of CO_2 per kWh

Country	CO_2 Factor
Sweden	0.04
France	0.09

Country	CO_2 Factor
Finland	0.24
Italy	0.59
Germany	0.60
Ireland	0.70
Luxembourg	1.08

IEA

Because a piece of equipment might be manufactured in one country, used in another, and dismantled in a third, the gain α is related to the stage of the life cycle (α_m will hence correspond to the factor during the manufacturing stage). The estimate of CO_2 pollution is then obtained by:

$$\Phi = \Phi_m + \Phi_u + \Phi_d = \alpha_m E_m + \alpha_u E_u + \alpha_d E_d \quad \textbf{(3.5)}$$

Moreover, Equation (3.5) addresses the location of energy production in order to use energy produced locally, thereby limiting energy transport losses. To integrate this parameter, it is necessary to make visible that part of the CO_2 emission caused by energy transportation. Therefore, a factor τ_s is added to Equation (3.5), giving:

$$\Phi = \sum_{s=m,u,d} \frac{\alpha_s}{\tau_s} \times E_s \quad \textbf{(3.6)}$$

The factor τ_s represents the additional rate for transporting energy at a stage s; for example, in France, the average energy lost during transport is 5% (i.e., $\tau_s = 0.95$).

Moreover Equation (3.6) can be extended to include other local sources of energy, such as oil-burning power generators (to power GSM antennas in an emergency) and solar panels installed locally and dedicated to the network. The impact depends on the nature of the energy (electricity, oil) so that the general metric for CO_2 emission becomes:

$$\Phi = \sum_{s=m,u,d} \sum_{\text{energy source } i} \frac{\alpha_{s,i}}{\tau_{s,i}} \times E_{s,i} \qquad (3.7)$$

where $E_{s,i}$ represents energy-source component i consumed during the manufacturing, use, or dismantling stages.

Equation (3.7) is suitable in the context of smart microgrids (Perea et al., 2008) whose the objective is to encourage the consumption of renewable energy produced locally.

Ethics in ICT

The design of green ICT solutions should integrate all pillars of sustainable development: planet, profit, and people. Mainly, the papers on green ICT focus on the environment and neglect to consider human and ethical aspects. Nevertheless, the basis of ICT development is to help the citizens in their daily life and in their working activities. Moreover, Herold (2006) explains that "the consideration of computer ethics fundamentally emerged with the birth of computers." The use of the people pillar is to cover many areas and employ subjective metrics, making the assessment difficult. In general, each ICT company should analyze its social responsibilities to its customers, employees, and community (Uddin et al., 2008).

Social responsibilities toward the customers: The ICT company must develop safe and durable ICT products and services, offer efficient after-sale services, and be prompt, reliable, and courteous in dealing with queries, complaints, and so on (Uddin et al., 2008). The company must not exaggerate the performance of proposed solutions, but should propose alternative solutions by clearly explaining its reasons for offering multiple choices to customers, and so on (Grupe et al., 2002). Rules regarding privacy and anonymity should be clearly defined to prevent the installation of cookies to obtain customer (or personal) data to sell to other companies (Herold, 2006). The company must also inform customers about the content of applications and any restrictions in regard to the law (for example, parental controls for videogames).

Social responsibilities toward employees: Ethics must be analyzed during the design process of each ICT project. The company must define a general code of conduct for the management of employees (salary, gender, productivity, etc.) and for the selection of subcontractors according to the consideration of the working conditions of their employees (exploitation of children, no salary, etc.). Moreover, during the design process, the company must respect simple and evident rules related to good practices in ICT such as not using unlicensed software to develop new software; it also must respect the private lives of employees based on e-mail exchanges and so on (Grupe et al., 2002).

Social responsibilities to the community: Ethics must also be considered before an ICT project is started. A preliminary study should analyze the impact of the newly developed system on society and individuals and whether it increases the digital divide and the social divide. According to Herold (2006), the issues regarding computers in the workplace require assessing the impact of ICT on the elimination of jobs, loss of skills, health issues, and computer crime in installing spyware, hacking and so on. Finally, social responsibilities to the community include educating ICT users to respect the environment, develop good new habits in turning off equipment not being used, turn off mobile phones in meetings, recycle ink cartridges, so forth.

To summarize, ICT companies should define ethical guidelines in order to produce ICT-based solutions in compliance with the requirements of the people pillar of sustainable development. Implicitly, an ethical approach has a positive impact on the choices realized in the planet pillar (environmental ethic) and in the profit pillar in promoting the fair business.

Conclusion

The assessment of green ICT solutions must consider many performance indicators gathering both numerical and subjective information about specific equipment. Many initiatives specify metrics that are either general (outside ICT scope) or specific to one ICT domain such as data centers. This section described three concrete MoPs that necessitate refinement and completion with new ones to cover all aspects of sustainable development. In waiting for standardized metrics, the design of ICT-based solutions must follow systemic approaches such as systems engineering. The interest of systems engineering is to handle complex systems, to specify measurable requirements in order to translate them into MoPs and to ensure that the performances are satisfied during all the project life cycle.

Systems Engineering for Designing Sustainable ICT-Based Architectures

Introduction

The development of green ICT solutions is highly complex because they must be based on an analysis of the system as a whole including the effects on business, the environment, and the people in the design loop during its entire life cycle. Therefore, the design of green ICT systems requires formal or semiformal methodologies and tools to specify, verify, and validate measures of performances according to both the user's requirements and sustainable development. Moreover, to reach the global green ICT objectives, the assessment should focus not only on the ICT performance of a system of interest but also on its environmental impact. In this section, the goal is to present an example for studying ICT systems in using systems engineering (Pyster et al., 2011; INCOSE, 2010). This example of a student project in the Erasmus Mundus Master in Pervasive Computing and Communications (PERCCOM) for sustainable development (www.perccom.eu) highlights some major issues of the requirements specification process (Jin, 2006).

Stakeholder Requirements Definition

The will-to-be operational domain in this study is an ICT architecture composed of ten computers and three Cisco 3560 PoE-24 ports switches. Two more reliable alternative solutions are also analyzed (Figure 3.3). The first one implements 4 switches and the second one 5 switches. The ICT architecture is planned for five years (use step). The ICT products are manufactured in China and the architecture is installed in France.

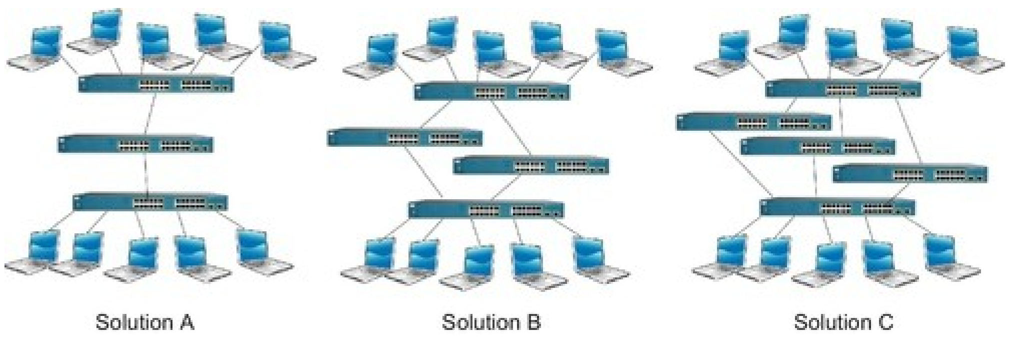


FIGURE 3.3 ICT-based architecture.

The objective of the system engineering process (Figure 3.4) is to analyze the performances of these ICT architectures (GICT0) regarding the pillars of ecology (GICT0.1), ethics (GICT0.2), and economics (GICT0.3) from the three ICT architecture solutions. In the ecology pillar, the intermediate requirements are to estimate the recyclability of ICT devices (GICT0.1.0), the radio wave emission (GICT0.1.1) and the ICT carbon emission (GICT0.1.2). In the ethic pillar, the requirement is to control anonymity and privacy of ICT users with appropriate data access policy. In the economy pillar, the requirement is to calculate the energy cost of ICT architecture.

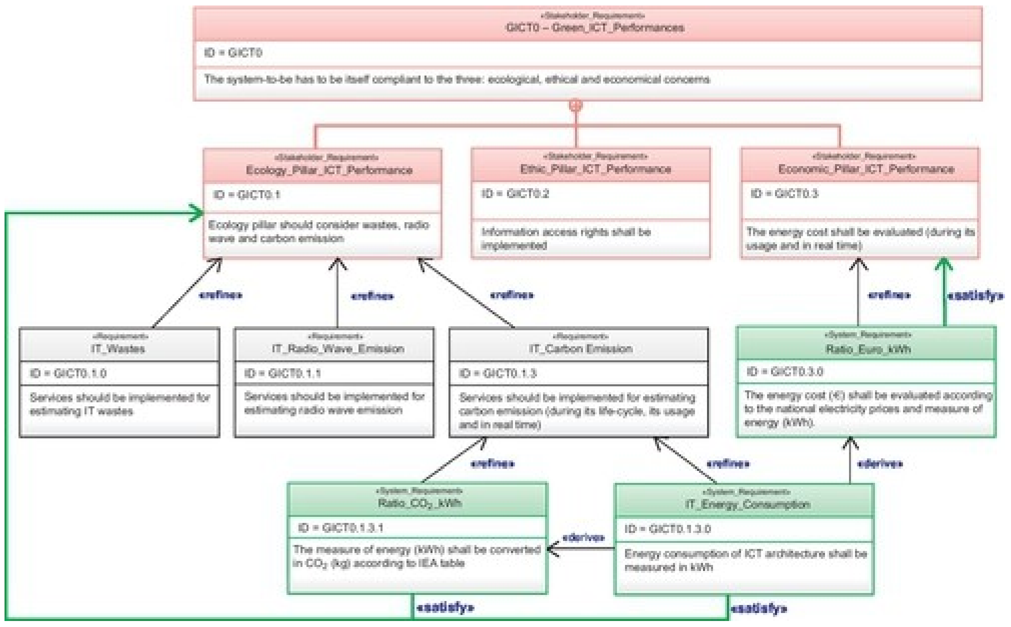

«Stakeholder_Requirement»
GICT0 – Green_ICT_Performances
ID = GICT0
The system-to-be has to be itself compliant to the three: ecological, ethical and economical concerns
«Stakeholder_Requirement»
Ecology_Pillar_ICT_Performance
ID = GICT0.1
Ecology pillar should consider wastes, radio wave and carbon emission
«Stakeholder_Requirement»
Ethic_Pillar_ICT_Performance
ID = GICT0.2
Information access rights shall be implemented
«Stakeholder_Requirement»
Economic_Pillar_ICT_Performance
ID = GICT0.3
The energy cost shall be evaluated (during its usage and in real time)
«refine»
«refine»
«refine»
«refine»
«satisfy»
«Requirement»
IT_Wastes
ID = GICT0.1.0
Services should be implemented for estimating IT wastes
«Requirement»
IT_Radio_Wave_Emission
ID = GICT0.1.1
Services should be implemented for estimating radio wave emission
«Requirement»
IT_Carbon Emission
ID = GICT0.1.3
Services should be implemented for estimating carbon emission (during its life-cycle, its usage and in real time)
«System_Requirement»
Ratio_Euro_kWh
ID = GICT0.3.0
The energy cost (€) shall be evaluated according to the national electricity prices and measure of energy (kWh).
«refine»
«refine»
«derive»
«System_Requirement»
Ratio_CO₂_kWh
ID = GICT0.1.3.1
The measure of energy (kWh) shall be converted in CO₂ (kg) according to IEA table
«derive»
«System_Requirement»
IT_Energy_Consumption
ID = GICT0.1.3.0
Energy consumption of ICT architecture shall be measured in kWh
«satisfy»
«satisfy»

FIGURE 3.4 System requirements of SysML diagram.

System Requirements Analysis

This chapter analyzed only the estimations of carbon emission and energy cost. Thus, system requirements (Figure 3.4) defined by the expert in systems engineering are specified in order to satisfy the stakeholder requirements. In both cases, the ICT energy consumption should be measured (GICT0.1.3.0). The carbon emission is calculated from the ICT energy consumption and the conversion tables between CO_2 and energy (GICT0.1.3.1). The energy costs are calculated from ICT energy consumption and the electricity price (GICT0.3.0).

The expert in systems engineering has no competence in the ICT domain to propose ICT solutions regarding the system requirements. As stated by Bouffaron et al. (2014), the driver of any specification process (as a requirement analysis process) is a quest for knowledge as design property from any specialist engineers to the system engineer. The aim of this knowledge is to bring the gap between stakeholder requirements, system, and ICT architectural constraints. Thus, ICT experts are involved in the design process to provide their knowledge in applying the green ICT MoPs defined in Section “Main Green Measures of Performances.” Figure 3.5 shows the interactions of all the knowledge domains of the study.

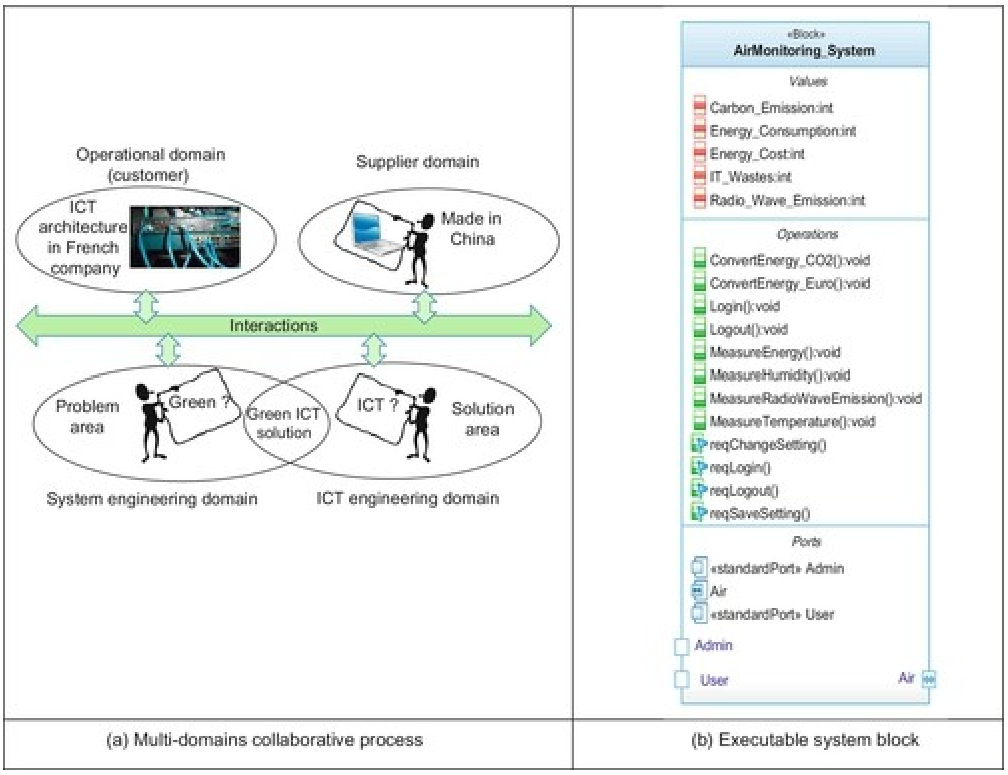


FIGURE 3.5 Systems specification process.

System Requirements Validation and Verification

The systems engineering project framework is based on systems thinking (Lawson et al., 2014) and on a model-based integrative approach aiming to check the “right-system requirements-right” from the early stages of a project. The goal is to explore in depth the problem to design first the overall required system in a concurrent, recursive, and iterative process in contrast to traditional sequential engineering approaches focusing first on solution issues. The use of models for verification and validation of measurable requirements to component solution integration is compliant with the last recommended best practices in industry (Fanmuy et al., 2012). The objective is to check the compliance of system requirements to stakeholder’s requirements by execution of models to functionally design the required system as a whole before sending it to an architect by allocating commercial off-the-shelf (COTS) or specific components. PERCCOM students execute their system specification by models transformed with a SysML-based tool (Sysml-Harmony) and are trained for ongoing developments related to *tool independent exchange of simulation models* (Blochwitz et al., 2012).

ICT Expertise and Results

The ICT expert proposes methods and results for providing the required information on the three ICT architecture solutions.

Function: energy consumption estimation of ICT architecture: The per hour energy consumption of all devices implemented in the ICT architecture has been measured by ICT experts; the values obtained are 35 Wh for the switch and 100 Wh for the computer. The network works 24/7 and the ten computers 10 h per day. By using the Equation (3.2) (E_u), the total energy consumptions of ICT architectures during five years are:

• Solution A: 22,849 kWh

• Solution B: 24,382 kWh

• Solution C: 25,915 kWh

Function: energy cost estimation of ICT architecture: With an average price of French kWh (0.12€ kWh), the total ICT costs of energy consumptions are:

• Solution A: 2741€

• Solution B: 2925€

• Solution C: 3109€

Function: carbon emission estimation of ICT architecture: By using the CO_2 factor per kWh in France (0.09), the ICT carbon emissions are [$\alpha_u E_u$ in Equation (3.5)].

• Solution A: 2056 kg $C0_2$

• Solution B: 2194 kg CO_2

• Solution C: 2332 kg CO_2

Function: energy consumption of ICT architecture life cycle: The analysis of energy consumption during the entire life cycle requires knowing the energy used for manufacturing and dismantling. According to EuPs (2007), the energy consumed to manufacture a computer is 900 kWh, and as approximation this value is used for computers and switches of architectures. The literature does not currently provide information about dismantling. Thus, 200 kWh is the empirical value selected.

By using the formula: $E_m + E_u + E_d$ (Equation (3.2)), the energy consumptions of ICT architectures are then:

• Solution A: 11,700 + 22,849 + 2600 = 37,149 kWh

• Solution B: 12,600 + 24,382 + 2800 = 39,782 kWh

• Solution C: 13,500 + 25,915 + 3000 = 42,415 kWh

Function: carbon emission of ICT architecture life cycle: Because the product is manufactured in China with a CO_2 factor per kWh at 0.97 and recycled in France, the carbon emissions are from the expression $\alpha_m E_m + \alpha_u E_u + \alpha_d E_d$ (Equation (3.5)):

• Solution A: (11,700 × 0.97) + (22,849 × 0.09) + (2600 × 0.09) = 11,349 + 2056 + 234 = 13,639 kg CO_2

• Solution B: (12,600 × 0.97) + (24,382 × 0.09) + (2800 × 0.09) = 12,222 + 2194 + 252 = 14,668 kg CO_2

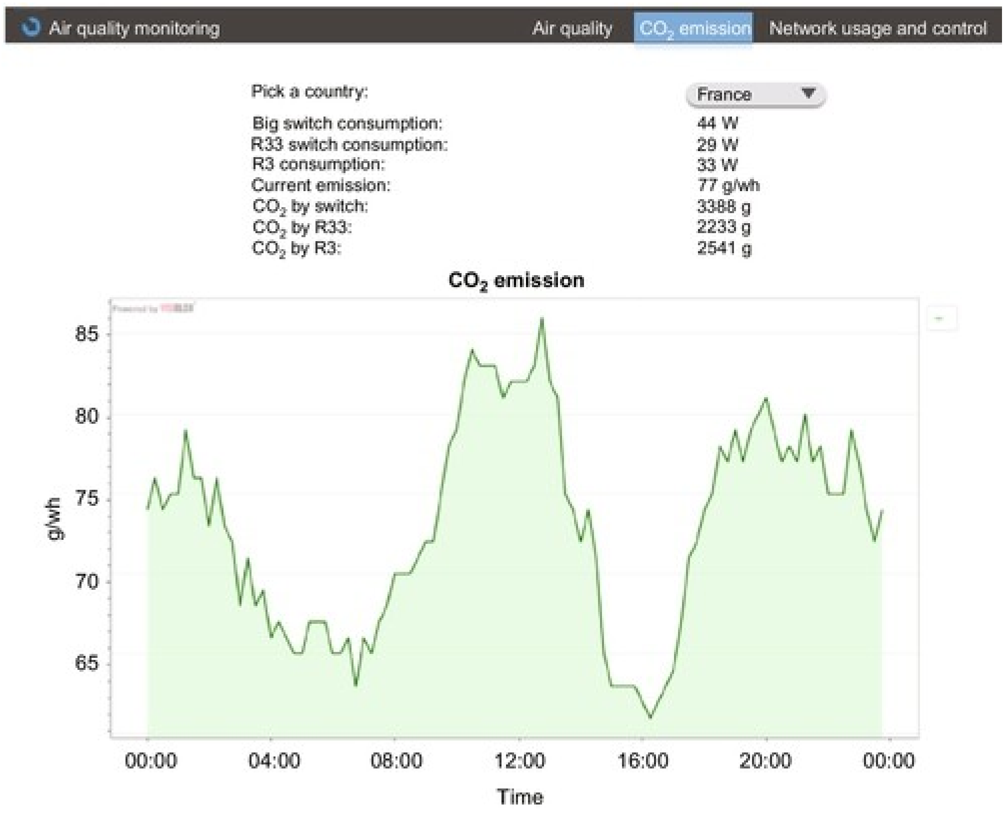
• Solution C: (13,500 × 0.97) + (25,915 × 0.09) + (3000 × 0.09) = 13,095 + 2332 + 270 = 15,697 kg CO_2

The interest of this result is not the global quantity of CO_2 emission but the ratio between the manufacturing and use steps. The carbon emitted during the manufacturing corresponds around to three years of carbon emission. In the context of the planet pillar, it appears that the engineering effort should be mainly achieved at the manufacturing level, not the use level. Nevertheless, the strong increase of energy prices is a good motivation to develop sophisticated engineering in order to mitigate the energy costs (profit pillar).

Function: real-time measure of ICT energy consumption: The network architecture is based on Cisco products to monitor energy consumption of devices implementing EnergyWise MIB. A monitoring software was developed for collection by using SNMP all energy consumptions of devices compatible with EnergyWise. The interest in using SNMP is to gather other ICT information such the bandwidth use, the temperature of equipment, and so on, and to easily propose indicators of efficiency, such as the ratio between energy consumption of ICT architecture and its use, to create smart monitoring in analyzing the variation of energy consumption regarding the temperature, traffic, and network configuration (number of connected ports, port speed, etc.).

Raritan intelligent PDUs or the ICT devices that do not implement EnergyWise are used. The Raritan PDU especially enables collection of the energy consumption of computers in the platform.

Function: real-time measure of ICT carbon emission: Carbon emission is estimated from the measurement of energy consumed by ICT architecture and from the CO_2 factor per kWh produced in France provided in real time by RTE and from the Internet. Figure 3.6 shows the interface developed in this project by PERCCOM students.



Pick a country:	France
Big switch consumption:	44 W
R33 switch consumption:	29 W
R3 consumption:	33 W
Current emission:	77 g/wh
CO_2 by switch:	3388 g
CO_2 by R33:	2233 g
CO_2 by R3:	2541 g

FIGURE 3.6 Monitoring ICT energy consumption and carbon emission.

Traceability Matrix

The design process of green ICT architectures must guarantee the sustainability of proposed solutions. The verification step is then crucial to check the process design. Table 3.3 shows that all functions were tested, confirming that the system requirements satisfy the stakeholder requirements. The study of traceability matrix avoids delivering ill-conceived projects by reconsidering the project. A redesign step should change the initial choices in selecting new ICT products to substitute nonadapted products, to develop software patches, to dismantle certain solutions, and so on. All these modifications generate premature and useless waste, consume additional energy, and generate wobbly, less sustainable solutions.

Table 3.3

Traceability Matrix

Stakeholder Requirements	System Requirements	Functions	Verification Method	Test Case	Test Result
GICT0.1 Ecology_Pillar_ICT_Performance	GICT0.1.3.0 IT_Energy_Consumption	Energy consumption estimation of ICT architecture	Documentation	TC 1	OK
		Energy consumption of ICT architecture life cycle	Documentation	TC 2	OK
		Real-time measure of ICT energy consumption	Demonstration	TC 3	OK
	GICT0.1.3.1 Ratio_CO2_Kwh	Carbon emission estimation of ICT architecture	Documentation	TC 1	OK
		Carbon emission of ICT	Documentation	TC 2	OK

Stakeholder Requirements	System Requirements	Functions	Verification Method	Test Case	Test Result
		architecture life cycle			
		Real-time measure of ICT Carbon emission	Demonstration	TC 3	OK
GICT0.3 Economic_Pillar_ICT_Performance	GICT0.3.0 Ratio_Euro_Kwh	Energy cost estimation of ICT architecture	Documentation	TC 1	OK

In conclusion, systems engineering provides good practices for the ecodesign of complex systems, especially to green the design of ICT projects.

Ecoefficiency Metrics

The results of the study described has obvious conclusions: Carbon emission and cost energy increase with the number of ICT products in the architecture. However, as Ascierto (2011) explained concerning PUE metric about data centers, the ICT system should be assessed in a global way when considering the intrinsic performances of ICT including the metrics presented in Section "ICT Technical Measures." For illustrating this comment, three stakeholder requirements are added in the study.

During the use of ICT architecture, the carbon emission must be less than 3000 kg CO_2, the energy cost must be less than 3000€, and the reliability must be SIL2 (10^{-7} < failures per hour < 10^{-6}) (IEC Standard 61508, 2005).

Thus, ecoefficiency metrics coupling the green performances with other intrinsic ICT performances should be developed. These metrics bring together various kinds of information, making it difficult to define a global equation that includes all requirements. A radar diagram is an interesting representation to show both the network performance and the requirements. Each MoP then corresponds to one axis of the radar. To improve the radar readability, all axes are normalized and the interest points emphasized. The global view offered by the radar diagram should make the analysis of network solutions easier, both for the designer and the customer. The best solution is the one in which all MoP results are in the center of the radar, but the best one relative to the requirement (optimized solution) is where the MoP results are close and beneath the specification line.

For estimating reliable MoP, it is necessary to define n_i as being the number of items of equipment forming a path *i*. Item is either a link or a switch. Let λ be the failure probability per hour for any item and $\mu = 1 - \lambda$ as the nonfailure probability. It is considered that all switches and links have failures per hour equal to $\lambda = 10^{-5}$. The failure probability of a network (*P*) composed of *p* independent paths depends on the failure probability of each path *i* (π) such that:

$$P = \prod_{i=1}^{p} P_i = \prod_{i=1}^{p} \left(1 - \mu^{n_i}\right)$$

For example, considering solution C, three paths are formed by three items (one switch plus two links). This gives:

$$P = \left(1 - \left(1 - 10^{-5}\right)^{3}\right)^{3} = 2.7 \times 10^{-14}$$

The results are for solution A: 3×10^{-5} and for solution B: 9×10^{-10}.

All MoP and requirements are collected in a single radar diagram (Figure 3.7). The system engineer can then select the most appropriate solution. The radar diagram shows that only solution B satisfies all stakeholder requirements (green line) because the reliability of solution A is not acceptable and the energy cost of solution C is too high. If all solutions are unsatisfactory, the stakeholder requirements should be redefined, refined in an iterative way with the system and ICT engineers to converge to consensual and acceptable solutions.

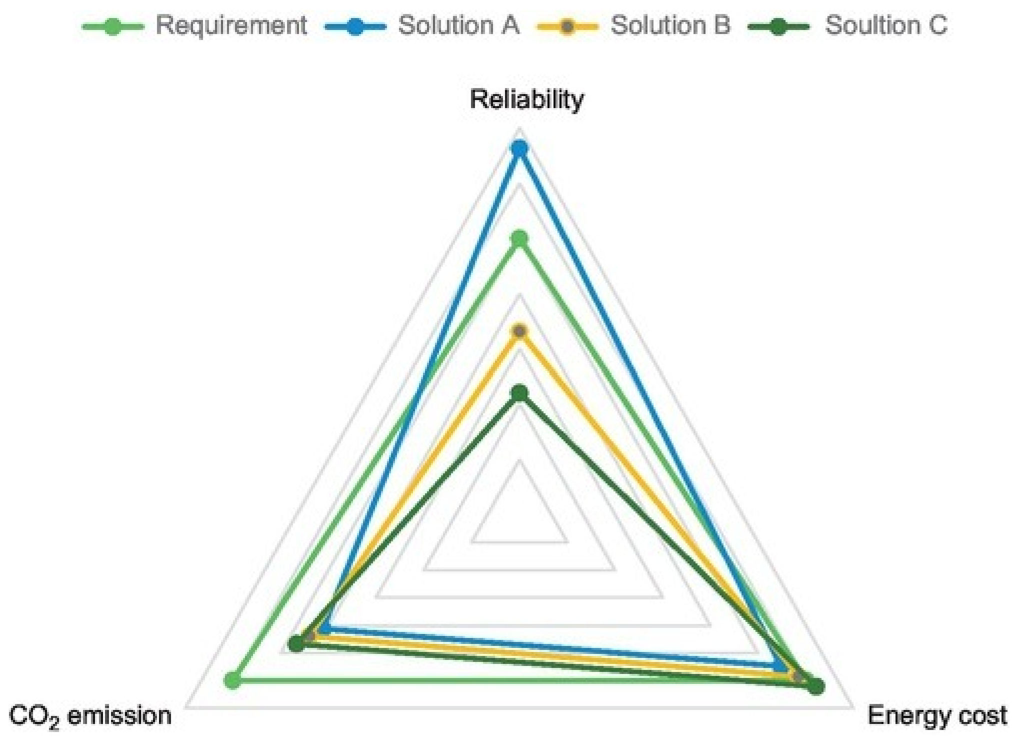


FIGURE 3.7 Radar diagram.

Regarding the ethical consideration, this iterative process in the systems engineering approach provides professionalism and respects the ethical bases for IT decision making defined by Grupe et al. (2002).

Conclusion

The design of green ICT-based solutions should consider the three pillars of sustainable development (Figure 3.1). The analysis is complex because it develops ICT solutions with the three objectives sometimes having conflicting criteria. Moreover, the analysis of the impact on the planet covers an unlimited scope that should include all pollution forms and the Earth's resource management, making a global assessment difficult during the entire ICT product life cycle. Pragmatically, green ICT mainly focuses on energy optimization. The major explanation is that it is a win-win activity because of its benefits on both planet and profit pillars. The recycling has also a positive result on both pillars because it enables the indefinite reuse of rare materials that are becoming more and more expensive. However, its implementation is very complex, and the return on investment is less immediate than energy. Regarding the people pillar, there are many contributions in the literature on computer ethics. However, from our best of knowledge, no significant contributions couple computer ethics with green ICT in a global sustainable development approach. People pillar should be analyzed at two levels: (1) how the ICT solution is designed and (2) how the developed ICT solution is used.

Nevertheless, the representation of sustainable development from a triangle is restrictive and especially hides the ICT performances corresponding to the fundamental activities of ICT engineers. Perhaps it is one factor explaining why the first green metrics in data centers do not include data center activities. The goal of the sustainable development triangle is to force ICT engineers to analyze the interactions between the three pillars to find balanced solutions that satisfy all stakeholders. However, the base requirement for an ICT professional when creating new ICT solutions is to specify SLA with economy and ICT metrics. In Figure 3.8, a new pillar is added, enabling the integration of ICT metrics in green ICT systems engineering. From this green ICT metric pyramid, the ICT engineer can study the relationships between ICT metrics and the pillars of sustainable development triangle in order to propose ecoefficient green ICT metrics and to specify green SLA.

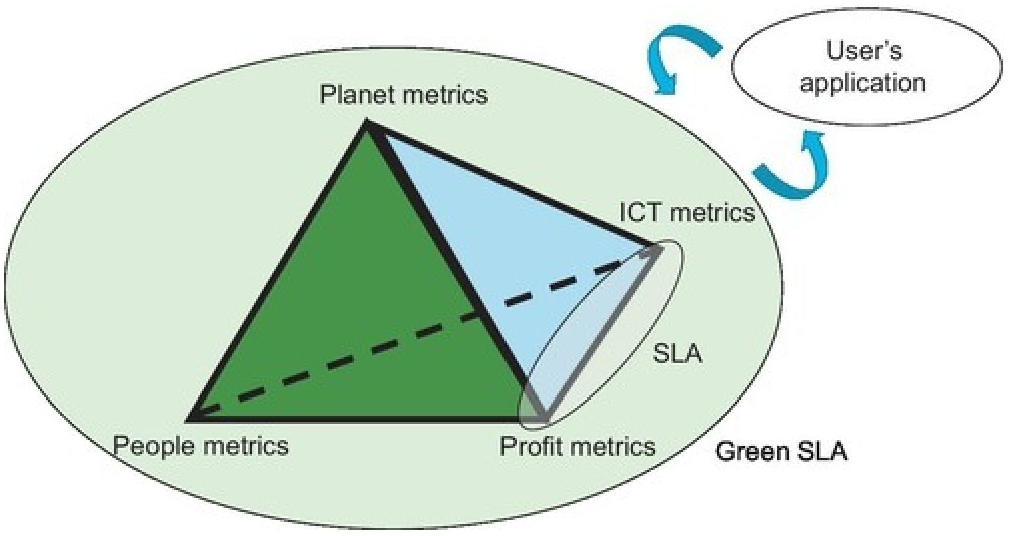


FIGURE 3.8 Green ICT metric pyramid.

A systems engineering approach for green ICT projects is recommended for developing more sustainable ICT-based solutions gathering in the same study all stakeholder requirements and enabling verification of the conformity of solutions. Moreover, new constraints directly related to climate change make ICT engineering more complex. Indeed, the ICT sector contributes to climate change, so it must propose resilient ICT solutions to climate changes. ITU (2014) explains the risks posed by both acute weather events (e.g., short-term extreme events such as cyclones, intense rainfall, and flooding) and chronic trends (e.g., long-term changes in temperature, seasonality, and the rise of sea levels) to guarantee reliable telecommunication infrastructures. For example, the installation of antenna should be more robust to withstand extreme wind, lightning, and heavy snowfall. A checklist of new constraints is proposed to ensure that all these extreme climate conditions were envisaged in ICT engineering step (ITU, 2014).

Finally, the SMART 2020 (Smart 2020, 2008) report shows the ambivalence of the ICT sector because it was responsible in 2007 for 2% of global carbon emissions and participates in climate change, but the sector could contribute to mitigating the carbon footprint of other human activity sectors related to transport, buildings, power, and industry by improving their energy efficiency. In this context, the role of ICT engineers is to design and create new smart metering in order to analyze and control the energy consumption of applications in real time. These metrics defined by applications (transport, building, etc.) are not included in the green ICT metrics presented in this chapter.

References

Allenby BR, Cooper WE. Understanding industrial ecology from a biological systems perspective. *Environ. Qual. Manage.* 1994;1520–6483(3):343–354.

Ascierto R. *From Money Pit to Profitability: The Business Case for Data center Efficiency.* 2011. http://ovum.com/research/from-money-pit-to-profitability-the-business-case-for-data-center-efficiency-metrics/.

Benyus JM. *Biomimicry: Innovation Inspired by Nature.* New York: Harper Perennial; 2002.0060533226.

Blochwitz T, Otter M, Akesson J, Arnold M, Claub C, Elmqvist H, Friedrich M, Junghanns A, Mauss J, Neumerkel D, Olsson H, Viel A. *Functional mockup interface 2.0: the standard for tool independent exchange of simulation models.* In: 9th International Modelica Conference, Munich, Germany, September 2012; 2012.

Bouffaron F, Dupont J-M, Mayer F, Morel G. *Integrative construct for model-based human-system integration: a case study.* In: Paper Presented at the World IFAC Congress 2014, Cape Town, South Africa; 2014.

Cerri G, De Leo R. Base-station network planning including environmental impact control. *IEE Proc. Commun.* 2004;22(2004):197–203.

Chapman PM. Determining when contamination is pollution—weight of evidence determinations for sediments and effluents. *Environ. Int.* 2007;33(4):492–501.

Cohen D. Earth's natural wealth: an audit. *New Sci. Mag.* (2605):2007;34–41.

Cui S, Goldsmith AJ, Bahai A. Energy-efficiency of MIMO and cooperative MIMO techniques in sensor networks. *IEEE J. Sel. Areas Commun.* 2005;22:1089–1098.

Diederen AM. *Metal Minerals Scarcity: A Call for Managed Austerity and the Elements of Hope.* Rijswijk, The Netherlands: TN Defence, Security and Safety; 2009.

Donough W, Braungart M. *Cradle to Cradle: Remaking the Way We Make Things.* first ed. New York: North Point Press; 2002.0865475873.

Drouant N, Rondeau E, Georges JP, Lepage F. Designing green network architectures using the ten commandments for a mature ecosystem. *Comput. Commun.* 2014;42:38–46.

Emerson Network Power, 2009. Energy logic: reducing data center energy consumption by creating savings that cascade across systems. A White Paper from the Experts in Business-Critical Continuity.

Encyclopedia Britannica, 2012. http://www.britannica.com/.

EuPs, 2007. European Commission DG TREN Preparatory Studies for Eco-design Requirements of EuPs. ISSN: 1404-191X.

European Commission Report, 2010. Water Scarcity and Drought in the European Union, KH-30-09-180-EN-D.

Fanmuy G, Fraga A, Llorens J. Requirements verification in the industry. In: *Complex Systems Design & Management.* Springer; 2012:145–160.

Foll LS. *TIC et Énergétique: techniques d'estimation de consommation sur la hauteur, la structure et l'évolution de l'impact des TIC en France.* (Ph.D. thesis) Institut national des télécommunications; 2008.

Georges JP, Divoux T, Rondeau E. Confronting the performances of a switched ethernet network with industrial constraints by using the network calculus. *Int. J. Commun. Syst.* 2005;18(9):877–903.

Graedel TE, Allwood J, Birat JP, Buchert M, Hagelüken C, Reck BK, Sibley SF, Sonnemann G. What do we know about metal recycling rates? *J. Ind. Ecol.* 2011;15(3):355–366.

Green Grid, 2012. PUE™: a comprehensive examination of the metric. White Paper 49.

Green Peace International Report, 2012. How is Clean Your Cloud? JN 417.

Grupe FH, Garcia-Jay T, Kuechler W. Is it time for an IT ethics program? *Information Management Strategy Systems and Technologies.* New York: Auerbach Publication; 2002.

GS1. *Implementation of GS1 EPC Global Standards in the Consumer Electronics Supply Chain.* 2010. http://www.gs1.org/.

Herold R. Introduction to computer ethics. *Information Systems Security.* New York: Auerbach Publications; 2006.

IEC Standard 61508, 2005. International Electrotechnical Commission, Functional Safety of Electrical/Electronic/Programmable Electronic Safety-related Systems.

IEEE, 2005. IEEE Standard for Safety Levels with Respect to Human Exposure to Radio Frequency Electromagnetic Fields, 3 kHz to 300 GHz, IEEE Std C95.1.

IEEE P1888, 2011. IEEE Standard for Ubiquitous Green Community Control Network Protocol, Ubiquitous Green Community Control Network Working Group, UGCCNet.

INCOSE. *Systems Engineering Handbook.* San Diego, CA: International Council on Systems Engineering; 2010.

ITU, 2011. Universal Power Adapter and Charger Solution for Mobile Terminals and Other Hand-held ICT Devices, Recommendation ITU-T L.1000.

ITU, 2014. Resilient Pathways: The Adaptation of the ICT Sector to Climate Change.

ITU-E800. *Terms and Definitions Related to Quality of Service and Network Performance Including Dependability.* 1994. http://www.itu.int/rec/T-REC-E.800.

ITU-T report, 2012. Toolkit on Environmental Sustainability for the ICT Sector, Chapter “General Specifications and Performances Indicators”.

ITU-T L.1001, 2012. External Universal Power Adapter Solutions for Stationary Information and Communication Technology Devices.

Jin Z. Revisiting the meaning of requirements. *J. Comput. Sci. Technol.* 2006;21:32–40.

Kesler SE. Mineral supply and demand into the 21st century. In: Briskey JA, Schulz KJ, eds. *Workshop on Deposit Modeling, Mineral Resource Assessment, and Sustainable Development.* U.S. Geological Survey Circular 1294; 2007:55–62 (paper 9).

Kubler S, Voisin A, Derigent W, Thomas A, Rondeau E, Främling K. Group fuzzy AHP approach to embed relevant data on “communicating material”. *Comput. Ind.* 2014;64(4):675–692.

Laszewski G, Wang L. *Green IT Service Level Agreements, Grids and Service-Oriented Architectures for Service Level Agreements.* New York: Springer Science; 2010 pp. 77-88.

Lawson B, Wade J, Hofkirchener W. *Systems Series Publishes Books Related to Systems Science, Systems Thinking, Systems Engineering and Software Engineering.* 2014. http://www.collegepublications.co.uk/systems/syt/.

Lewis A. *The Four Key Environmental Factors of ICT: Energy, Carbon, E-Waste and Water.* 2013. http://www.sustainability-perspectives.com/perspective/four-key-factors.

Makela T, Luukkainen S. Incentives to apply green cloud computing. *J. Theor. Appl. Electron. Commer. Res.* 2013;8(3):74–86.

Michaut F, Lepage F. Application oriented network metrology: metrics and active measurement tools. *IEEE Commun. Surv. Tutorials Second Quart.* 2005;7(2):2–24.

Perea E, Oyarzabal JM, Rodríguez R. Definition, evolution, applications and barriers for deployment of microgrids in the energy sector. *Elektrotech. Informationstech.* 2008;125:432–437 0932-383X.

Pyster A, Olwell D, Anthony J, Enck S, Hutchison N, Squires A. *A Guide to the Systems Engineering Body of Knowledge (SEBoK) version 0.75.* Hoboken, NJ: Stevens Institute of Technology; 2011.

Reviriego P, Sivaraman V, Zhao Z, Maestro J, Vishwanath A, Sanchez-macian, Russell C. *An energy consumption model for energy efficient ethernet switches.* In: 10th Annual International Conference on High Performance Computing and Simulation, HPCS 2012, Madrid, 2-6 July 2012; 2012:98–104.

RICS QS & Construction Standards, 2012. Methodology to Calculate Embodied Carbon of Materials, P 32/2012, UK.

Ritz T, Thalau P, Phillips JB, Wiltschko R, Wiltschko W. Resonance effects indicate a radical-pair mechanism for avian magnetic compass. *Nature.* 2004;429(6988):177–180.

Seitz H, Stinner D, Eikmann T, Herr C, Roosli M. Electromagnetic hypersensitivity (EHS) and subjective health complaints associated with electromagnetic fields of mobile phone communication—a literature review published between 2000 and 2004. *Sci. Total Environ.* 2005;349:45–55.

Smart 2020, 2008. http://www.smart2020.org.

Stevenson C, Chouinard G, Lei Z, Hu W, Shellhammer S, Caldwell W. IEEE 802.22: the first cognitive radio wireless regional area network standard. *IEEE Commun. Mag.* 2009;47(1):130–138.

Tuppen C. *Circularity and the ICT Sector, Ellen MacArthur Foundation/Advancing Sustainability LLP.* Nancy: PERCCOM Seminar; 2013. www.perccom.eu/files/2013/09/Nancy-ICT-CE-copy.pdf.

Uddin MB, Hassan MR, Tarique KM. Three dimensional aspects of corporate social responsibility. *Daffodil Int. Univ. J. Bus. Econ.* 2008;3(1):199–212.

Val-Europe, 2012. http://www.valeurope-san.fr/info/UK/00.

Vatanski N, Georges JP, Aubrun C, Rondeau E, Jämsä Jounela SL. Networked control with delay measurement and estimation. *J. Control Eng. Pract.* 2009;17(2):231–244.

WEEE. *Waste Electrical and Electronic Equipment, Directive 2002/96/EC of the European Parliament and of council of 27 January 2003.* Bruxelles: Official Journal of the European Union; 2003.

World Health Organization, 2006. Base Stations and Wireless Technologies, Fact Sheet No. 304.

World Health Organization, 2011. Electromagnetic Fields and Public Health: Mobile Phones, Fact Sheet No. 193.

Williams ED, Sasaki Y. *Energy Analysis of End-of-life Options for Personal Computers: Resell, Upgrade, Recycle.* In: Proceedings of the 2003 IEEE International Symposium on Electronics and the Environment; New Jersey: IEEE: Piscataway; 2003:187–192.

CHAPTER 6

Sustainable Cloud Computing

Konstantinos Domdouzis Sheffield Hallam University, Sheffield, UK

Abstract

Sustainability is an important subject that is part of worldwide policy agendas for economic, social, and environmental transformation. Solutions, especially for environmental issues, must be found and necessary measures be taken to address them. Cloud computing is a new technology that has been developed in order to provide a new era of applications, which will allow more efficient use of computing power. In this chapter, a thorough introduction to cloud computing is realized with emphasis on its advantages for environmental sustainability. A list of challenges in relation to the use of the technology as green technology is presented, and the reasons for using cloud computing for sustainability are explained. Finally, a detailed list of the applications of cloud computing focusing on social, business, and environmental sustainability are listed, and a number of conclusions are provided.

Keywords

Sustainability

Cloud

Environment

Research

Governance

Energy

Virtualization

SaaS

PaaS

IaaS

Grid

Introduction

Globalization has resulted in the development of new opportunities in the process of economic growth and has led to the resolution of many social problems. New businesses are created and social issues are reported in a matter of seconds. However, this progress has negative impacts. The most important consequences of globalization can be shown to be on the natural environment. Some decades ago, environmental issues were considered only as possible future problems, but today they are recognized as current issues as governments try to take measures or explore environmentally friendly technologies to stop the constant environmental disaster. The creation of the World Wide Web and the Internet was the driving force behind globalization. These two technologies managed to transform the entire planet into a digital village and created new routes for scientific research and knowledge. However, the consequences of digital technologies were also negative. The maintenance of a huge number of computer servers and the daily use of millions of personal computers have contributed to the increase in the

Earth’s carbon footprint. New technologies, such as cloud computing, have emerged and provide hope that computing power can be harnessed so that it does not affect the environment.

Clouds include more than the Internet. According to Mell and Grance (2011), cloud computing is a model for enabling a shared pool of computing resources on demand that can be rapidly released with minimal management effort. Dikaiakos et al. (2009) define *cloud computing* as the transfer of information technology (IT) services from the desktop and into large data centers. Qian et al. (2009) have defined the architecture of cloud computing used in organizations as one with two main layers, the core stack and the management layer. The core stack layer includes three sublayers: applications, platform, and resource (Figure 6.1).

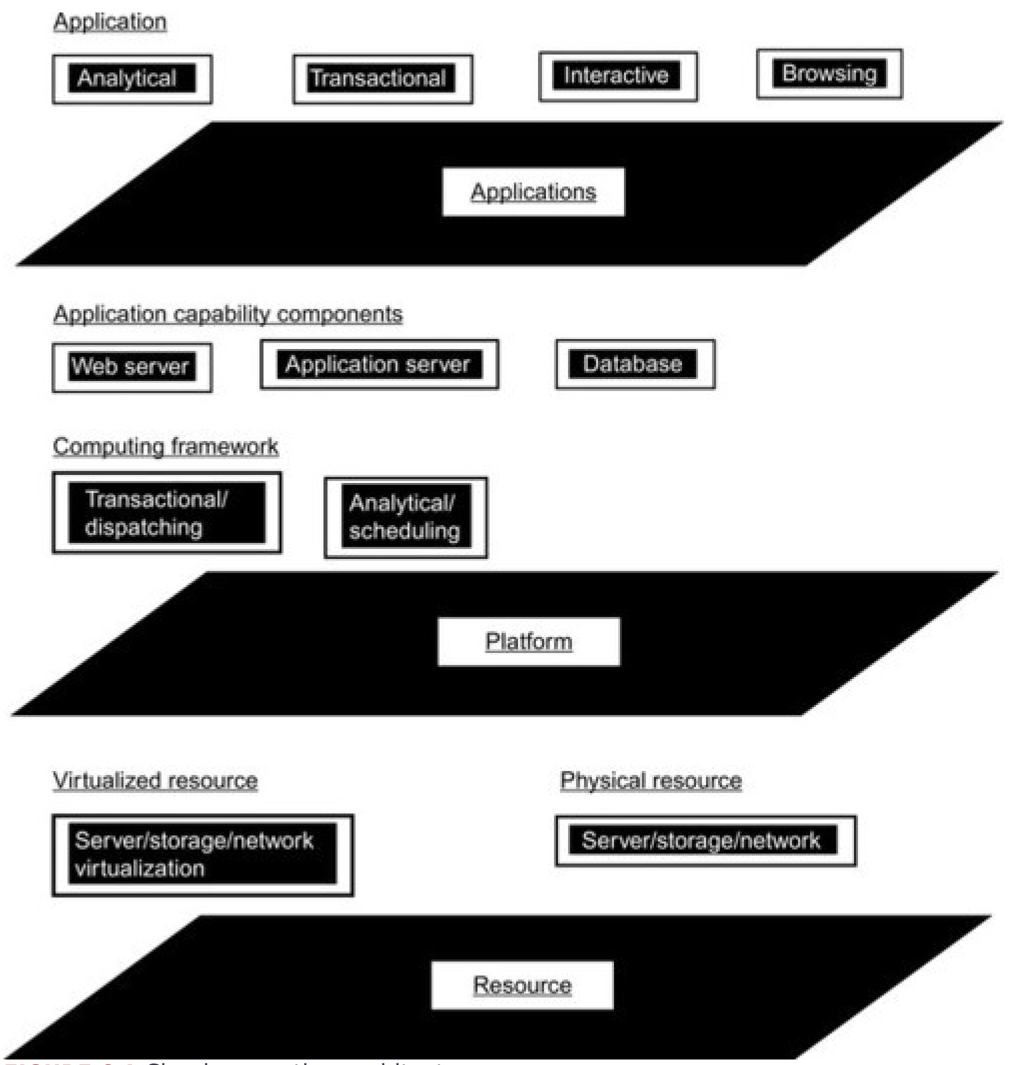


FIGURE 6.1 Cloud computing architecture. Source: Adapted from Qian et al. (2009).

There are different categories of cloud computing, such as hardware as a service (HaaS), software as a service (SaaS), platform as a service (PaaS), and infrastructure as a service (IaaS). HaaS allows users to rent an entire data center on a pay-as-you-go subscription. SaaS hosts a software application that customers can use without installing the software on their local computers (Wang and von Laszewski, 2008). Cloud systems can provide the software platform on which to run a system (PaaS) (Vaquero et al., 2009). With infrastructure as a service (IaaS), the user can deploy and run his or her own operating system in addition to the virtualization software offered by the provider (Prodan and Ostermann, 2009). Table 6.1 lists some technologies for the different cloud computing platforms.

Table 6.1

Cloud Computing Platforms

Cloud Computing Framework	Example Technology
SaaS	Google Apps, Facebook, YouTube
PaaS	MS Azure, Google AppEngine, Amazon SimpleDB/S3
IaaS	(Infrastructure) Amazon EC2, GoGrid, Flexiscale
	(Hardware) Data Centers
HaaS	Servers, backup

The advantages of using cloud technologies are numerous. They provide flexibility by providing IT services, such as software updates and troubleshooting security issues, and allow the development of shared applications, thus promoting online collaboration. Cloud computing does not require the use of high-quality equipment and is easy to use. In addition, it enables the sharing of data between different platforms (Zhang et al., 2010). Cloud computing provides new horizons to business competitiveness because it enables small businesses to use new technology in order to compete more efficiently with larger competitors by using IT services more rapidly and in a more secure way. Through cloud computing, companies use the server space they need, therefore reducing the consumption of energy in comparison to that used by on-site servers.

Cloud computing can be considered an example of green IT. Jenkin et al. (2011) define *green ITs* as the technologies that directly or indirectly promote environmental sustainability in organizations. The term *sustainability* refers to the fair resource distribution between present and future generations and between agents in the current generation to maintain a scale of the current economy in relation to the ecological system that supports it (Costanza, 1994). The term *environmental* *sustainability* refers to the preservation of the natural environment's ability to support human life. This can be achieved only by making appropriate decisions. Environmental sustainability has a long history. In 1842, Jean Baptiste Joseph Fourier provided the first reference to greenhouse gases. In 1896, Svante Arrhenius suggested that levels of carbon dioxide in the atmosphere impact global temperatures via the greenhouse effect. The first major conference on environmental sustainability, the Stockholm Conference for the Human Environment, was held in 1972. In 1992, the United Nations Framework Convention on Climate Change (UNFCC) was held in Rio de Janeiro. The Kyoto protocol signed in 1997 legally binds 37 industrialized countries to specific measures (Boone and Ganeshan, 2012).

In order to achieve environmental sustainability, fundamental changes in production must be realized. These changes are parallel to the continuous evolution of technological systems, especially the creation of innovative technological systems. Hekkert et al. (2007) specify that the rate of technological change is specified by the competition among existing innovation systems, not only different technologies. In order to understand innovative systems, a definition of technological systems must be provided. Technological systems involve a network of agents that interact in the economic/industrial area under a particular institutional infrastructure and support the generation and production of technology (Carlsson and Stankiewicz, 1991). Innovative technological systems therefore must be used in order to achieve sustainability. They can be required to be sustainable by using technologies such as those that manage resources in such a way that performance and power are optimized. Additionally, they enable the realization of research with more efficient use of global energy sources. Cécile et al. (2002) specify that innovative policies for sustainable development should promote technological diversity and long-term innovation capacity with parallel use of clean technologies. Watson et al. (2010) mention that it is important to follow a scientific approach in the development of IT for environmental sustainability and to stop using approaches that involve management and policy formation (Watson et al., 2010).

Challenges in the Use of Cloud Computing As Green Technology

There are a number of challenges associated with the use of green cloud computing. The main challenge is to minimize energy use and satisfy the requirements related to quality of service. However there are a number of issues when cloud computing is considered for environmental applications:

• *Energy-aware dynamic resource allocation*: In cloud computing, the excessive power cycling of servers could negatively impact their reliability. Furthermore, in the dynamic cloud environment, any interruption of energy can affect the quality of the provided service. Also, a virtual machine (VM) cannot record the timing behavior of a physical machine exactly. This can lead to timekeeping problems and inaccurate time measurements within the VM, which can result in incorrect enforcement of a service-level agreement (SLA) (Kalange Pooja, 2013).

• *Quality of service (QoS)-based resource selection and provisioning*: QoS-aware resource selection plays a significant role in cloud computing. Better resource selection and provision can result in energy efficiency (Kalange Pooja, 2013).

• *Optimization of virtual network topologies*: Because of VM migrations or the machines' nonoptimized allocation, communicating VMs may ultimately be hosted on distant physical nodes, and as a result, the cost of data transfer between them may be high (Kalange Pooja, 2013).

• *Enhancing awareness of environmental issues*: The users of green cloud computing technologies should become aware of the use of the technology in the resolution of specific environmental problems, such as the reduction of carbon emissions.

• *SLAs*: These provide for the replication of one application to multiple servers. Cloud customers should evaluate the range of parameters of SLAs, such as data protection, outage, and price structure, offered by different cloud vendors (Padhy et al., 2011).

• *Cloud data management*: Cloud operation is characterized by the accumulation of large amounts of data. Cloud service providers rely on cloud infrastructure providers to achieve full data security. In addition, VMs can migrate from one location to another; therefore, any remote configuration of the cloud by service providers could be insufficient (Padhy et al., 2011).

• *Interoperability*: Many public cloud systems are closed and are not designed to interact with each other. Industry standards must be created in order to allow cloud service providers to design interoperable cloud platforms. The Open Grid Forum is an industry group that is working on open cloud computing interface to provide an application program interface (API) for managing different cloud platforms (Padhy et al., 2011).

• *Security*: Identity management and authentication are very significant, especially for government data. Governments, specifically the US government, have incorporated cloud computing infrastructures into the work of various departments and agencies. Because the government is a very complex entity, the implementation of cloud computing involves making policy changes, implementing dynamic applications, and securing the dynamic environment (Paquette et al., 2010).

Cloud Computing and Sustainability

When combined with specific characteristics of other technologies, such as the distributed resource provision of grid computing, the distributed control of digital ecosystems, and the sustainability from green computing, cloud computing can provide a sociotechnical conceptualization for sustainable distributed computing. Grid computing is a form of distributed technology in which a virtual supercomputer includes a cluster of networked computers performing very large tasks (Foster and Kesselman, 2004). Digital ecosystems are sociotechnical systems characterized by self-organization, scalability, and sustainability (Briscoe and De Wilde, 2006; Briscoe, 2009). Their purpose is to extend service-oriented architecture (SOA), thus supporting network-based economies (Newcomer and Lomow, 2005). Green computing is the efficient use of computing resources, which respects specific values for societal and organizational success. These values are people, planet, and profit (Marinos and Briscoe, 2009).

Social, economic, and environmental sustainability can also be achieved through the development of a green infrastructure (Benedict and McMahon, 2002). Weber et al. (2006) define *green infrastructure* as the abundance of landscape features that in combination with ecological processes contributes to human health. Lafortezza et al. (2013) identify a green infrastructure framework (GIF) that includes five functions: ecosystem services, biodiversity, social and territorial cohesion, sustainable development, and human well-being. These elements interact with each other

A number of cloud computing business models have been developed to ensure sustainability. Examples are the cloud cube model that enables collaboration in cloud formations used for specific business needs and the hexagon model, which provides six main criteria (consumers, investors, popularity, valuation, innovation, and get the job done [GTJD]) for business sustainability and shows how cloud computing performs according to these criteria (Chang et al., 2010). The cloud can lead to business sustainability through business improvement and the transformation and creation of new business value chains (Berman et al., 2012).

Focal companies are responsible for the supply chain; they provide direct contact to the customer. Additionally, they design the product or service offered. Focal companies are also responsible for the environmental and social performance of their suppliers; therefore, there is increased need for sustainable supply chain management (Seuring and Müller, 2008; Handfield and Nichols, 1999; Schary and Skjøtt-Larsen, 2001).

There is a direct linkage between suppliers, focal companies, and customers. The integration of environmental thinking into supply chain management results in green supply chain management. In this case, sustainable information systems can be used to provide sustainable information services in the supply chain. Research on green information systems can be classified to different categories. The first category involves the examination of how the software development life cycle can be modified to reduce the potential negative environmental impacts of systems (Haigh and Griffiths, 2008). The second involves research on environmental reporting, measurement and accounting systems, and the use of knowledge management for environmental sustainability. Some research studies are also related to the consideration of environmental parameters when designing new products. A cloud computing platform can be characterized as a green information system that provides green information services.

The sustainability offered by cloud computing can be shown by the fact that the specific technology allows small business organizations to access large amounts of computing power in a very short time; as a result, they become more competitive with larger organizations. Third-world countries can also be significantly benefit by cloud computing technologies because they can use IT services that they previously could not access because they lacked the resources. Cloud computing accelerates the time in the businesses market because it allows quicker access to hardware resources without any up-front investment. In this case, there is no capital expenditure (capex), only operational expenditure (opex). Cloud computing makes possible the realization of new, innovative applications such as real-time, location-, environment-, and context-aware mobile interactive applications; parallel batch processing used for the processing of large amounts of data during very short periods of time; and business analytics for customer behavioral analysis (Marston et al., 2011).

Sustainable Applications of Cloud Computing

The architecture, engineering, and construction (AEC) sector is a highly fragmented, project-based industry with very strong data sharing. Beach et al. (2013) describe how cloud computing can be used in the AEC sector for better data management and collaboration. In order to create an efficient architecture for a cloud computing prototype, the authors used a building information model (BIM) data representation that is a complete 4-D virtual repository of all the data related to a construction project, such as 3-D models of building structure, construction data management information such as plans and schedules, information about all items within the building, and data about the progress of the construction project. The specific cloud computing platform is based on CometCloud, which is an autonomic computing engine for cloud and grid environments. Specifically, it is useful in the development of clouds with resizable computing capability that both integrates local computational environments and public cloud services and enables the capability to develop a range of programming applications (Kim and Parashar, 2011). The CloudBIM prototype was built using CometCloud’s master/worker programming model and includes three main elements: client, masters, and workers. The function of CloudBIM is based on the interaction between masters and workers. The workers store data and are responsible for the validation of each query they receive. The client is responsible for the provision of the interface between the users and the local master node. This interface converts users’ actions into queries (Beach et al., 2013).

Social sustainability can also be achieved through the use of social networks. This type of network can be based on cloud platforms or cloud applications and can be realized in social networks. The creation of social networking sites provides easier and less expensive ways for sustainable development communities to develop a wide variety of new communities. Examples of social networking sites for social sustainability are TakingITGlobal, which supports youth-led action using blogs, online groups and event calendars, UnLtd that supports social entrepreneurship, and People for Earth, a social network that provides advice on how people should live more environmentally friendly lives.

Busan is South Korea’s second largest city and the fifth largest container globally. A cloud infrastructure has been provided to Busan based on Cisco Unified Computing System. The developed solution is the Busan Smart + Connected Communities, which aims to deliver social, economic, and environmental sustainability. It connects the Busan metropolitan government, the Busan Mobile Application Centre, and five local universities. During 2014, it is expected that the Busan cloud platform will create 3500 new job opportunities and 300 start-up companies focused on the Mobile Application Center development (Cisco, 2012).

Cloud computing has been also applied to government. The US government has made efforts to introduce cloud computing to the General Services Administration (GSA), the National Aeronautics and Space Administration (NASA), the Department of the Interior, the Department of Health and Human Services (HHS), the Census Bureau, and the White House. Specifically, the GSA can provide cloud-based hosting of the federal government's primary e-government portals—USA.gov—and its Spanish-language companion site, GobiernoUSA.gov. NASA uses the Nebula cloud computing platform, which can provide transparency in the involvement with space efforts. The US Department of the Interior's National Business Center (NBC) has introduced several cloud-based human resources management applications, including Web-based training, staffing, and recruitment programs. The NBC also offers cloud-based financial and procurement software (U.S. Department of the Interior, NBC, 2009). The Program Support Center (PSC) of the Department of HHS has selected Salesforce.com for the SaaS pilot, and within weeks, it had a working pilot online (Gross, 2009). The US Census Bureau uses Salesforce.com's SaaS to manage the activities of about 100,000 partner organizations. The White House uses also cloud computing to allow US citizens to express their opinions about President Obama (Hart, 2009). The UK government has also introduced the use of cloud computing through the development of G-cloud, which is a governmentwide cloud computing network (Glick, 2009). Petrov (2009) identified efforts from specific European countries, such as Sweden, France, and Spain, focused on the implementation of cloud computing for economic development, health and education services, and transportation networks.

Cloud computing is currently used for the advancement of scientific research. An example is the Cumulus project, which is an ongoing cloud computing project established at the Steinbuch Centre for Computing of the Karlsruhe Institute of Technology (KIT). The aim of Cumulus is the provision of virtual infrastructures for scientific research (Juve and Deelman, 2010). Cloud computing is extremely significant for the European Organization for Nuclear Research (CERN), which realizes a number of experiments (e.g., the ALPHA experiment, the Isotope mass separator on-line facility [ISOLDE] experiment, the large Hadron collider (LHC), the total elastic and diffractive cross-section measurement) and generates amazingly extreme amounts of data (approximately 15 petabytes) useful for simulation and analysis. The CERN Data Centre is responsible for the collection of these data. In 2002, the CERN Data Centre required the use of the worldwide LHC computing grid (WLCG), which is a distributed computing infrastructure in tiers that provides to 8000 physicists real-time access to LHC data. However, CERN is currently focused on the use of cloud technology for better handling these colossal data sets (CERN, 2014). It is very probable that CERN will adopt the use of the OpenStack cloud computing software in order to update its data management operations (Purcell, 2014). Other examples of the use of cloud computing for scientific research are Red Cloud from Cornell University, Wispy from Purdue University, and the San Diego Supercomputer Center (SDSC) cloud (University of Chicago and Argonne National Laboratory, 2014).

Global Forest Watch is an online mapping tool created by Google, the World Resources Institute, and 40 other partners. It is based on the analysis of Landsat7 satellite images that are processed by Google Earth Engine, the company’s cloud platform for geodata analysis. The amount of image data that were processed was about 20 tera-pixels, which required 1 million central processing unit (CPU) hours on 10,000 computers operating in parallel for a period of several days. Google estimates that a single computer would need 15 years to perform the tasks realized by Global Forest Watch. The aim of the mapping system is also the enhancement of satellite images with social data, which will document possible forest abuse (Claburn, 2014).

Cloud computing is used to process an inversion process for magnetotellurics, a geophysic technique used for the characterization of geothermal reservoirs and mineral exploration. The entire inversion process is implemented on the Amazon Elastic Compute Cloud (EC2) cloud using available EC2 instances The determination of subsurface electrical conductivity is significant in a range of applications covering tectonic evolution and mineral and geothermal exploration. Mudge et al. (2011) describe how a Fortran program was developed in order to abstract logic and process modeling calculations. The program is embedded in a web application. The whole inversion process is implemented on the Amazon EC2 cloud using available EC2 instances. The whole process is further packaged as a MT 3D inversion software product, which is accessible anywhere through a secure login. The magnetotellurics method is used for understanding the processes involved during the Enhanced Geothermal Systems (EGS) fluid system.

Lawrence Berkeley National Laboratory and Northwestern University in the United States created a modeling tool that shows the energy savings from moving local network software and computing to serve cloud farms. The specific tool is called Cloud Energy and Emissions Research Model (CLEER). The aim of the model is the provision of a framework for the assessment of the net energy consumption and greenhouse gas emissions of cloud computing in comparison to traditional systems (Irfan and ClimateWire, 2013). Cloud computing saves energy through virtualization, which allows virtual machine consolidation and migration, heat management, and temperature-aware allocation; these are techniques that result in the reduction of power consumption (Garg and Buyya, 2012). The large-scale deployment of virtualized server infrastructure can result in the balance of computer and storage loads across physical servers. The optimized design of cloud data centers allows servers to run at optimal temperature and use. Furthermore, cloud computing provides dynamic provisioning of infrastructure capacity and sharing of application instances between client organizations that results to the reduction of peak loads (accenture and WSP, 2010).

Eye on Earth is an environmental information sharing system developed by the European Environment Agency (EEA), which is based on Microsoft Windows Azure cloud platform. Information sharing is important in efficiently and effectively addressing environmental issues. The users of the environment can be policy makers, communities or individuals, environmental organizations, and emergency responders. Different environmental data, such as air and water quality are examined and can be represented in a visual format. For example, an intelligent map allows users to discover environmental information (European Environment Agency, 2014).

The US government has launched the Climate Change Initiative to help organizations and citizens to use public data more efficiently to better prepare for the effects of climate change. Google has donated 50 million hours of cloud computing time on the Google Earth Engine geospatial analysis platform to the project. Google Earth Engine can detect changes on Earth's surface using satellite imagery. Examples of its application are the development of the first global maps of deforestation and a nearly real-time system that identifies deforestation resulting from climate change (Google, 2014). The engine will be used to manage the agricultural water supply and model the impacts of sea level rises (Weiss, 2014).

Information and communication technologies (ICTs) play an important role in emergency response systems during natural disasters. Alazawi et al. (2011) suggest the use of an intelligent disaster management system based on the use of vehicular ad-hoc networks (VANETs) and mobile and cloud technologies. VANETs are the most advanced technology of intelligent transportation systems (ITS). They are vehicles that are equipped with sensors and wireless communication capability. The main focus of the application of VANETs is safety and transportation efficiency. The cloud-based vehicular emergency response system includes three main layers: the cloud infrastructure, the Intelligence, and the system interface layers. The cloud infrastructure layer is the foundation for the base platform for the emergency system. The intelligence layer provides the necessary computational models to develop optimum emergency response strategies. The system interface layer collects data from the Internet, social media, and even roadside masts. The system was implemented in Ramadi, Iraq.

Suakanto et al. (2012) suggest the use of cloud computing infrastructure for detecting environmental disasters early and monitoring environmental conditions. Depending on the type of environmental application in which the system is used, different sensors are used. For example, for air monitoring applications, carbon dioxide or air quality control sensors are used; for water monitoring applications, water pH measurement sensors can be used. A remote terminal unit (RTU) is used in order to collect data from the different types of sensors in analogue and digital form. The units are placed in areas that are prone to disasters. A cloud platform is used for central data storage and processing (Figure 6.2).

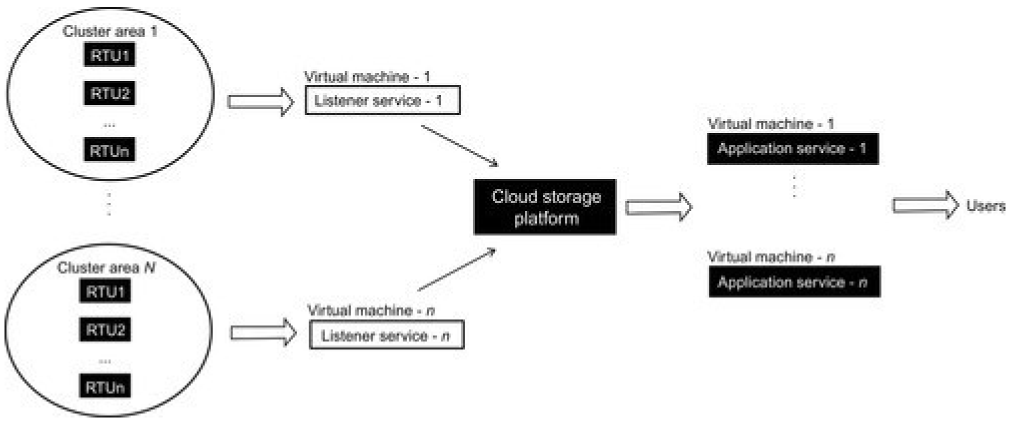


FIGURE 6.2 Cloud platform for environmental situations. Source: Adapted from Suakanto et al. (2012).

Data collected by the sensors are sent to the listener service of the virtual machines. The main function of this service is to read and process sensor data and store them in the cloud storage repository. When users are online, they can see the stored data by using application services installed on virtual machines.

Agricultural modernization has great significance for China's growing economy. Even though the country has realized improvements in crop cultivation and animal and plant breeding, agricultural production is a decentralized operation characterized by the use of low-level information technologies. Agricultural modernization involves the use of modern agricultural equipment, modern agricultural planting and breeding technology, and modern forms of production organization and management. Cloud computing can lead to the establishment of an information network services platform and the integration of isolated production facilities, technical equipment, and information services. Applications of cloud computing are the integration and sharing of agricultural information, the real-time monitoring of agricultural production, the provision of agricultural technology services, construction and improvement of agricultural supply chain, and the tracking of the quality of agricultural products (Zhu et al., 2013).

Technologies Associated With Sustainable Cloud Computing

Some technologies have the same characteristics as cloud computing or are related to cloud computing are:

• *Grid computing*: This involves a large network of computers used to create large supercomputing resources. In these types of networks, large and complex computing operations can be performed.

• *Virtualization*: Virtualization allows the consolidation of multiple applications into virtual containers located on a single or multiple servers (Padala et al., 2007). Virtualization is the foundation of cloud computing. It enables users to access servers without knowing details of those servers.

• *Utility computing*: This term defines a “pay-per-use” model for using computing services.

• *Autonomic computing*: The term *autonomic* refers to self-management, in this case, of computers. In autonomic computing, computers can correct themselves without human intervention.

• *Web services*: Web services simplify the application-to-application interaction using a systematic framework based on existing Web protocols and open XML standards. This framework includes three mail elements: communication protocols, service descriptions, and service discovery (Curbera et al., 2002).

Future Prospects of Sustainable Cloud Computing

The future of cloud computing in relation to its impact to sustainability is excellent. There is huge potential for additional applications of the technology to different industries, such as manufacturing, health care, and education. Examples of this potential are the provision of access to global data resources through cloud computing, the realization of low-cost simulation experiments, provision of massive and flexible computing power for drug discovery, and real-time health monitoring. With its rapid prototyping and collaborative design and the improvement of manufacturing processes, cloud computing can also contribute significantly to supply chain coordination. Furthermore, the technology can be the basis for highly interactive, collaborative learning (World Economic Forum, 2010).

The European Union (EU) envisions the realization of the global cloud ecosystem. It can offer new features to support cloud employment and to improve adoption of cloud computing. It could provide supporting tools that would cover issues related to building and supporting new platforms easily, new programming models, and tools that deal with distribution and control, improved security and data protection, efficient data management, energy efficiency, and easy mash-ups of clouds exposing a single-user interface. The EU can also exploit the capabilities offered by existing cloud systems to enhance the capabilities of products and services offered by European industry (European Commission, 2014). Developing countries, such as India, can seek ways to enhance sustainability through the use of green IT. In fact, India spent 18.2 billion for green IT in 2013. A study by Global e-Sustainability Initiative and Microsoft showed that running services over the cloud can be 95% more efficient than those run in other ways (Thomond, 2013).

Reflections on Sustainable Cloud Computing Applications

Sustainability is a topic of great importance regarding continuous environmental, social, and economic problems. Cloud computing can be applied to all aspects (social, business, environmental) of sustainability. Specific characteristics of the technology, such as sharing data, allow small businesses to have access to huge computing power and as a result, these businesses can become more competitive. Cloud technology can be adjusted to meet requirements of sustainability. It offers dynamic provisioning of resources, multitenancy (the serving of multiple businesses using the same infrastructure), server use, and the power efficiency of data centers (accenture and WSP, 2010). The technology has huge potential in shaping sustainability in different fields. Especially related to environment, it promotes environmental research, development of emergency response systems, and implementation measures to prevent climate change.

However, there are some concerns in relation to the energy efficiency of cloud computing. When files to be transmitted over a network are quite large, the network will be a major contributor to energy consumption. Furthermore, the VM consolidation could reduce the number of active servers but will put an excessive load on few servers whose heat distribution can become a major issue. A green cloud framework that considers these problems can be developed. In such a framework, users submit their cloud service requests through a middleware green broker that manages the selection of the greenest cloud provider to serve the user's request.

A user request can be for software, platform, or infrastructure. The cloud providers register their services in the form of offers to a public directory and the green broker assesses them. These offers include green services, pricing, and time when the service should be assessed for the least carbon emission. The green broker calculates the carbon emission of all cloud providers. It then selects the services that will result in the least carbon emission and buys them on behalf of users. Some additional steps need to be taken: the design of software at various levels (OS, compiler, algorithm, application) that facilitates energy efficiency, the measurement of data center power, and the deployment of the data centers near renewable energy sources (Garg and Buyya, 2012).

Conclusions

Cloud computing is an advanced technology with great impact on different fields. Michael Dertouzos, late director of Massachusetts Institute of Technology Laboratory of Computer Science, had the vision to make computing an indispensable part of human life, just like oxygen. Therefore, he initiated Project Oxygen for the development of human-centered, pervasive computing. Cloud computing is the best example of such technologies. It has revolutionized the way services are offered through the World Wide Web by reducing the dependence to hardware. Most importantly, applications can run independently from specific computer configurations.

Cloud computing is a significant factor for achieving sustainability. It promotes academic research, the development of businesses, and the strengthening of business competition. Governmental agencies have started integrating the technology in their existing infrastructure, and social media have gained widespread acceptance. The impact that cloud computing can have in the future for the environment is significant. Applications such as the realization of emergency response systems during natural disasters and the early detection of environmental disasters can be efficiently realized through the use of cloud computing. Social networking can also be used for the development of teams that promote environmental consciousness or provide information that can be useful during environmental crises. However, more research needs to be conducted future to fully comprehend the energy efficiency provided by the use of this specific technology.

References

accenture and WSP, 2010. Cloud computing and sustainability: the environmental benefits of moving to the cloud. Technical report [online]. Available at: http://www.accenture.com/SiteCollectionDocuments/PDF/Accenture_Sustainability_Cloud_Computing_TheEnvironmentalBenefitsofMovingtotheCloud.pdf (accessed 01.03.2014.).

Alazawi Z, Altowaijri S, Mehmood R, Abdljabar MB. *Intelligent disaster management system based on cloud-enabled vehicular networks.* In: ITS Telecommunications (ITST) 11th International Conference; 2011:361–368. doi:10.1109/itst.2011.6060083.

Beach TH, Rana OF, Rezgui Y, Parashar M. Cloud computing for the architecture, engineering & construction sector: requirements, prototype & experience. *J. Cloud Comput. Adv. Syst. Appl.* 2013;2:8.

Benedict MA, McMahon ET. Green infrastructure: smart conservation for the 21st century. *Renewable Resour. J.* 2002;20:12–17.

Berman SJ, Kesterson-Townes L, Marshall A, Srivathsa R. How cloud computing enables process and business model innovation. *Strategy Leadersh.* 2012;40(4):27–35.

Boone T, Ganeshan R. By the numbers: a visual chronicle of carbon dioxide emissions. In: Boone T, Jayaraman V, Ganeshan R, eds. *Sustainable Supply Chains - Models Methods, and Public Policy Implications.* New York: Springer; 2012.

Briscoe G. *Digital Ecosystems.* Ph.D. dissertation London: Imperial College London; 2009.

Briscoe G, De Wilde P. *Digital ecosystems: evolving service-oriented architectures.* In: Conference on Bio Inspired Models of Network, Information and Computing Systems; Cavalese, Italy: IEEE Press; 2006. [online]. Available at: http://arxiv.org/abs/0712.4102.

Carlsson B, Stankiewicz R. On the nature, function and composition of technological systems. *J. Evol. Econ.* 1991;1(2):93–118.

Cécile, P., Valenduc, G., Warrant, F., 2002. Technological innovation fostering sustainable development. Levers for a Sustainable Development, Policy.

CERN, 2014. Computing-experiments at CERN generate colossal amounts of data. The Data Centre stores it, and sends it around the world for analysis [online]. Available at: http://home.web.cern.ch/about/computing (accessed 10.04.2014.).

Chang V, Wills G, De Roure D. *Cloud business models and sustainability: impacts for businesses and e-research.* In: UK e-Science All Hands Meeting 2010, Software Sustainability Workshop, Cardiff, GB, 13-16 Sep 2010; 2010:3.

Cisco, 2012. City transforms economic sustainability with public cloud, customer case study. Available at: http://smartcitiescouncil.com/system/tdf/public_resources/cisco_busan%20ec%20dev.pdf?file=1&type=node&id=247 (accessed 18.04.2014.).

Claburn. *Google Powers Forest Protection Effort.* 2014. [online]. Available at: http://www.informationweek.com/government/cloud-computing/google-powers-forest-protection-effort/d/d-id/1113950 (accessed 18.04.2014.).

Costanza R. Chapter 24: three general policies to achieve sustainability. In: Jansson A, Hammer M, Folke C, Costanza R, eds. *Investing in Natural Capital.* Washington, DC: Island Press; 1994.

Curbera F, Duftler M, Khalaf R, Nagy W, Mukhi N, Weerawarana S. Unraveling the web services web—an introduction to SOAP, WSDL, and UDDI. *IEEE Internet Comput.* 2002;6(2):86–93.

Dikaiakos MD, Katsaros D, Mehra P, Pallis G, Vakali A. Cloud Computing: Distributed Internet Computing for IT and Scientific Research. *IEEE Internet Comput.* 2009;13(5):10–13. doi:10.1109/MIC.2009.103.

European Commission. *The Future of Cloud Computing—Opportunities for European Cloud Computing Beyond 2010.* In: 2014. [online]. Available at: http://cordis.europa.eu/fp7/ict/ssai/docs/cloud-report-final.pdf (accessed 19.04.2014.).

European Environment Agency. *Eye on Earth Briefing.* 2014. [online]. Available at: http://www.eea.europa.eu/about-us/what/seis-initiatives/eye-on-earth-briefing/view (accessed 20.03.2014.).

Foster I, Kesselman C. *The Grid: Blueprint for A New Computing Infrastructure.* San Francisco, CA: Morgan Kaufmann; 2004.

Garg SK, Buyya R. Green cloud computing and environmental sustainability. In: Murugesan S, Gangadharan GR, eds. *Harnessing Green IT Principles and Practices.* Chichester, West Sussex: Wiley; 2012:315–337.

Glick B, Glick B. Digital Britain commits government to cloud computing. *Computing.* 2009. [online]. Available: http://www.computing.co.uk/computing/news/2244229/digital-britain-commits (accessed 15.06.2013.).

Google. *Helping Our Communities to Adapt to Climate change.* 2014. [online]. Available at: http://google-latlong.blogspot.co.uk/2014/03/helping-our-communities-adapt-to.html (accessed 14.04.2014.).

Gross, G., 2009. Gov't agencies embrace cloud computing: government agencies say they're moving toward an embrace of cloud computing and software-as-a-service. PC World, February 25, 2009. [online]. Available: http://www.pcworld.com/article/160233/article.html (accessed 22.02.2014.).

Haigh N, Griffiths A. The environmental sustainability of information systems: considering the impact of operational strategies and practices. *Int. J. Technol. Manage.* 2008;43(1/2/3):48–63.

Handfield RB, Nichols EL. *Introduction to Supply Chain Management.* New Jersey: Prentice-Hall; 1999.

Hart, K., 2009. Tech firms seek to get agencies on board with cloud computing. *The Washington Post*, March 31, 2009. [online]. Available at: http://www.washingtonpost.com/wp-dyn/content/article/2009/03/30/AR2009033002848.html (accessed 15.04.2014.).

Hekkert MP, Suurs RAA, Negro SO, Kulhmann S, Smits R.E.H.M. Functions of innovation systems: a new approach for analysing technological change. *Technol. Forecasting Social Change.* 2007;74:413–432.

Irfan, U., ClimateWire, 2013. Cloud computing saves energy. Sci. Am., June 12, 2013. [online]. Available at: http://www.scientificamerican.com/article/cloud-computing-saves-energy/ (accessed 22.04.2014).

Jenkin TA, Webster J, McShane L. An agenda for ‘Green’ information technology and systems research. *Inf. Organ.* 2011;21(1):17–40.

Juve G, Deelman E. Scientific workflows and clouds. *ACM Crossroads.* 2010;16(3):14–18.

Kalange Pooja R. Applications of green cloud computing in energy efficiency and environmental sustainability. *J. Comput. Eng. 1.* 2013;25–33.

Kim H, Parashar M. *CometCloud: An Autonomic Cloud Engine, Cloud Computing: Principles and Paradigms.* Wiley; 2011 (chapter 10), pp. 275-297.

Lafortezza R, Davies C, Sanesi G, Konijnendijk C. Green Infrastructure as a tool to support spatial planning in European urban regions. *iForest.* 2013;6:102–108.

Marinos A, Briscoe G. *Community cloud computing.* In: Proceedings of the 1st International Conference on Cloud, Computing; 2009:472–484.

Marston S, Li Z, Bandyopadhyay S. Cloud computing—the business perspective. *Decis. Support Syst.* 2011;51(1):176–189.

Mell, P., Grance, T., 2011. The NIST definition of cloud computing—recommendations of the national institute of standards and technology. National Institute of Standards and Technology, Special Publication 800-145.

Mudge JG, Chandrasekhar P, Heinson GS, Thiel S, Mudge JG, Chandrasekhar P, Heinson GS, Thiel S. *Evolving inversion methods in geophysics with cloud computing.* In: Proceedings of IEEE eScience 2011, Stockholm; 2011.

Newcomer E, Lomow G. *Understanding SOA with Web Services.* Upper Saddle River, NJ: Addison-Wesley; 2005.

Padala P, Zhu X, Wang Z, Singhal S, Shin KG. *Performance Evaluation of Virtualization Technologies for Server Consolidation.* 2007. [online]. Available at: http://137.204.107.78/tirocinio/site/tirocini/Tirocinio-Zuluaga/Documents/virtualizzazione/Technologies%20for%20Server.pdf (a ccessed 03.05.2014.).

Padhy RP, Patra MR, Satapathy SC. Cloud computing: security issues and research challenges. *Int. J. Comput. Sci. Inf. Technol. Secur.* 2011;1(2):136–146.

Paquette S, Jaeger PT, Wilson SC. Identifying the security risks associated with governmental use of cloud computing. *Gov. Inf. Q.* 2010;27:245–253.

Petrov, O., 2009. Backgrounder: financial crisis and cloud computing—delivering more for less. Demystifying cloud computing as enabler of government transformation. World Bank, Government Transformation Initiative, June 16, 2009. [Online]. Available at: http://www.siteresources.worldbank.org/.../BackgrounderFinancialCrisisCloudComputing.doc (accessed 11.04.2014.).

Prodan R, Ostermann S. *A survey and taxonomy of infrastructure as a service and web hosting cloud providers.* In: 10th IEEE/ACM International Conference on Grid Computing, October 13-15; 2009:17–25.

Purcell A. *Tim Bell on the Importance of OpenStack for CERN.* 2014. [online]. Available at: http://www.isgtw.org/feature/tim-bell-importance-openstack-cern (accessed 10.04.2014.).

Qian L, Luo Z, Du Z, Guo L. Cloud computing: an overview. *Cloud Comput. Lect. Notes Comput. Sci.* 2009;5931:626–631.

Schary P, Skjøtt-Larsen T. *Managing the Global Supply Chain.* second ed. Copenhagen: Copenhagen Business School Press; 2001.

Seuring S, Müller M. From a literature review to a conceptual framework for sustainable supply chain management. *J. Cleaner Prod.* 2008;16(5):1699–1710.

Suakanto S, Supangkat SH, Roberd Saragih S, Arief Nugroho T, Gusti Bagus I, Nugraha B, Suakanto S, Supangkat SH, Roberd Saragih S, Arief Nugroho T, Gusti Bagus I, Nugraha B. *Environmental and disaster sensing using cloud computing infrastructure.* In: International Conference on Computer and Communications Security—ICCCS; 2012:doi:10.1109/ICCCSN.2012.6215712.

Thomond P. *The Enabling Technologies of a Low-Carbon Economy: A Focus on Cloud Computing.* In: 2013. [online]. Available at: http://gesi.org/assets/js/lib/tinymce/jscripts/tiny_mce/plugins/ajaxfilemanager/uploaded/Cloud%20Study%20-%20FINAL%20report_2.pdf (accessed 20.04.2014.).

U.S. Department of the Interior, National Business Center (NBC), NBC's Federal Cloud Playbook, August 2009. [Online]. Available at: http://api.ning.com/files/Rn9mXJiYBdsR1iwPJ68dWWEwO6TEkRCO18Xu-Hj7QfIbqZ-*Oc4fswll48ShViNARu3RaoZ8vfUrRxe5SUKnMr7jcLSbQ6I2/NBCCloudWhitePaperFinalWebRes.pdf (accessed 01.04.2014.).

University of Chicago and Argonne National Laboratory. *Clouds.* 2014. [online]. Available at: http://scienceclouds.org/infrastructure-clouds/ (accessed 10.04.2014.).

Vaquero LM, Rodero-Merino L, Caceres J, Lindner M. A break in the clouds: towards a cloud definition. *ACM SIGCOMM Comput. Commun. Rev.* 2009;39(1):50–55.

Wang L, von Laszewski G. *Scientific cloud computing: early definition and experience.* In: 10th IEEE International Conference on High Performance Computing and Communications (HPCC '08), September 25-27, Dalian, China; 2008.

Watson RT, Boudreau M-C, Chen AJ. Information systems and environmentally sustainable development: energy informatics and new directions for the IS community. *MIS Quart.* 2010;34(1):23–38.

Weber T, Sloan A, Wolf J. Maryland's green infrastructure assessment: development of a comprehensive approach to land conservation. *Landscape Urban Plan.* 2006;77(1–2):94–110.

Weiss TR. *Google Provides Cloud Computing Resources for Climate Change Research.* 2014. [online]. Available at: http://www.eweek.com/cloud/google-provides-cloud-computing-resources-for-climate-change-research.html (accessed 15.04.2014.).

World Economic Forum. *Exploring the Future of Cloud Computing: Riding the Next Wave of Technology-Driven Transformation.* 2010. [online]. Available at: http://www.weforum.org/pdf/ip/ittc/Exploring-the-future-of-cloud-computing.pdf (accessed 15.04.2014.).

Zhang S, Zhang S, Chen X, Huo X, Zhang S, Zhang S, Chen X, Huo X. *Cloud computing research and development trend.* In: Second International Conference on Future Networks, (ICFN '10), January 22-24, 2010, Sanya, China; 2010:93–97.

Zhu Y, Wu D, Li S. Cloud computing and agricultural development of china: theory and practice. *IJCSI Int. J. Comput. Sci.* 2013;10(1):7–12.

CHAPTER 7

Sustainable Software Design

Michael Engel Leeds Beckett University, Leeds, UK

Abstract

The sustainability of software depends on direct effects, which result from executing programs, as well as indirect effects, which influence the sustainability of the product incorporating the software. Indirect effects are caused, e.g., by planned obsolescence when a device manufacturer stops to provide software updates, rendering the device useless.

In order to optimise direct effects, tools are required which provide methods for energy modelling and to analyse and optimize the energy consumption of a given program. Indirect effects are harder to assess, since their optimization requires changing economical incentives. However, it is important to optimise both kinds of effect in order to provide sustainable software, since the amount of energy consumed during the operation of a device is in the same order of magnitude as the energy used during its production.

This chapter gives an overview of the various parameters involved and provides links to publications discussing optimization approaches.

Keywords

Sustainability

Production

Environmental impact

Energy consumption

Moore's law

Life-cycle analysis

Overview and Scope

Sustainability is a broad term encompassing a large number of concepts. In the area of software engineering, a number of different definitions exist. For instance, Tate (2005) defines sustainable software development as follows:

Sustainable software development is a mindset (principles) and an accompanying set of practices that enable a team to achieve and maintain an optimal development pace indefinitely.

In the context of this chapter, however, the software development process plays only a minor part. Here, *sustainability* is used in an ecological context, as defined by Dick et al. (2010):

Sustainable Software is software, whose direct and indirect negative impacts on economy, society, human beings, and environment that result from development, deployment, and usage of the software are minimal and/or which have a positive effect on sustainable development.

Adhering to this definition, in the following text we analyze the direct as well as indirect effects that software has on the sustainability of combined hardware/software systems throughout the product life cycle.

Evaluating Sustainability Effects

In order to evaluate and compare the environmental impact of software, a common base must be established. In general, the overall environmental impact of a product is difficult to assess. Especially during production, a large number of factors influence this impact. During the production of LCD screens, for example, an inorganic compound, the gas NF_3 (nitrogen trifluoride), is used, which is estimated to be about 17,000 times more effective as a greenhouse gas than carbon dioxide (CO_2), according to Weiss et al.(2008). In order to achieve comparability between the impact that different chemical elements and processes have on the environment, most studies break this impact down to an equivalent of CO_2 impact, measured in kg of CO_2 emitted into the atmosphere.

However, there is no universally accepted method to assess the environmental friendliness of software. The related metrics found in current literature differ in terms of measures of resources, the kind of result generated, the analyzed environment and application domain, and their usage. Bozzelli et al. (2013) performed an extensive analysis of metrics used to assess the “greenness” of software. Their study concludes that most researchers exclusively concentrate on metrics related to energy consumption at runtime but neglect the production phase of the overall product. Nevertheless, this study is useful to obtain an overview of the different approaches and to assess the suitability of a given metric for a specific context.

Sustainability and the Product Life Cycle

An assessment of the sustainability properties of software must take place in the context of the product the software is intended for. As shown in Figure 7.1, the life cycle of all IT products consists of a number of stages, starting from the sourcing of raw materials and ending with the disposal or recycling of the product. For the impact analysis of software, this life cycle can roughly be categorized into three sources: the production—including the sourcing and processing of raw materials and its distribution to the end customer—use, and disposing or recycling of a product.

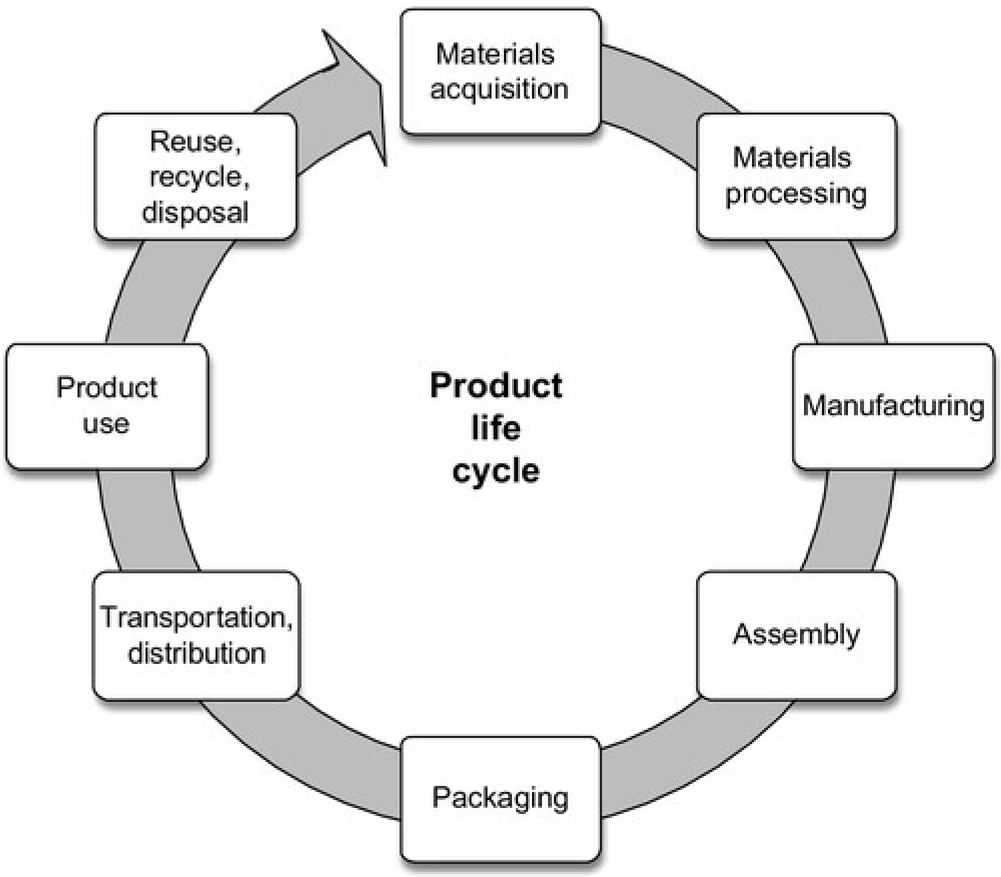


FIGURE 7.1 The life-cycle of IT products.

According to recent analyses by Marwedel and Engel (2010) and Deng et al. (2011), the manufacturing of current IT devices, such as laptop computers and flat-panel displays, plays an important role when considering a product's overall sustainability. Overall, manufacturing is responsible for up to 50% of the overall environmental impact of a product (when converting the impact to CO_2 impact), whereas the use of a product contributes largely to the remaining impact. Other influences, such as transporting a product from the factory to the customer or recycling a device, have only negligible impact. Figure 7.2 shows an example of the distribution of environmental impact over the life cycle of a typical desktop PC, assuming a typical lifetime of five years. Transportation and assembly make up only 6% of the overall environmental impact, whereas production and use generate the largest impacts. However, the environmental impact depends on additional factors, such as the energy mix used to power the operation of the devices. Figure 7.3 shows the different CO_2 impacts of operating a PC in a number of European countries with a different mix of energy sources.

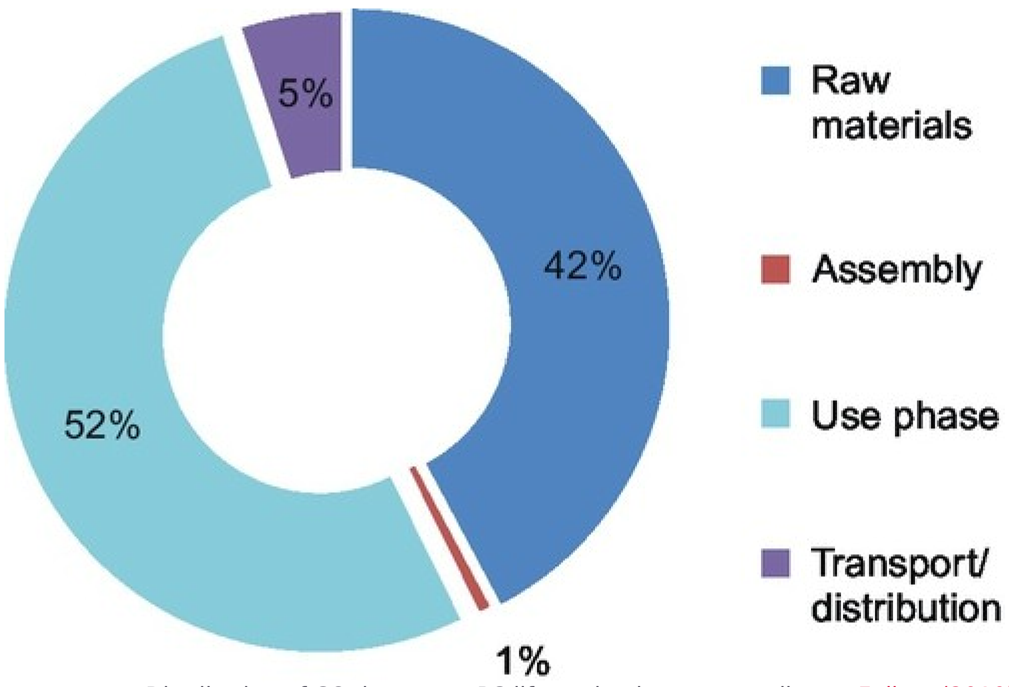


FIGURE 7.2 Distribution of CO_2 impact to PC life-cycle phases according to Fujitsu (2010).

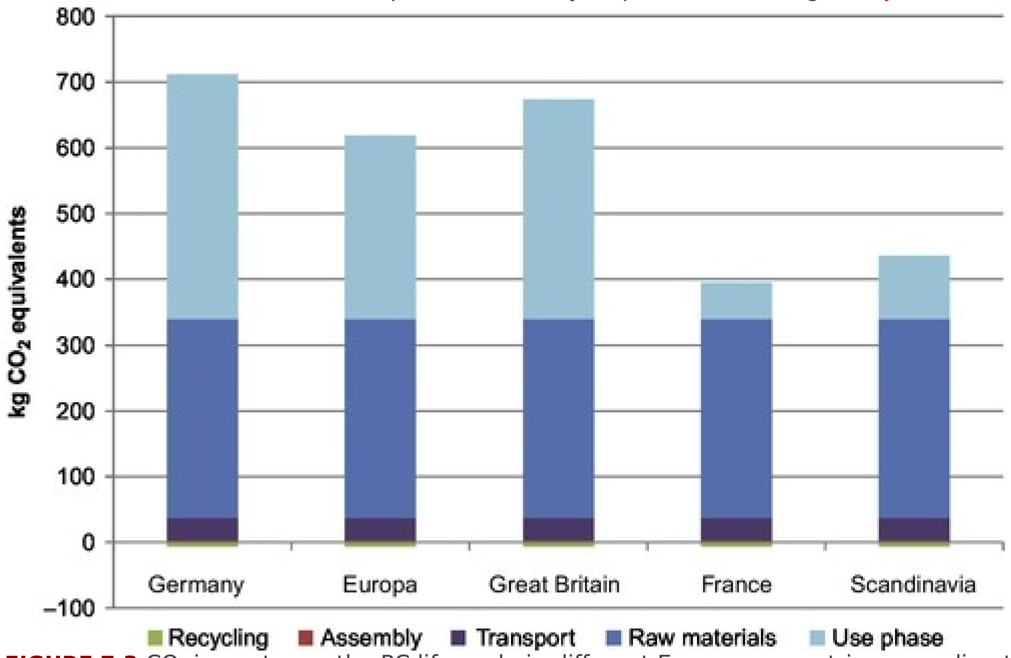


FIGURE 7.3 CO_2 impact over the PC life-cycle in different European countries according to Fujitsu (2010).

This distribution of environmental impact over a product's life cycle implies that two different means of impact of software products must be considered. On the one hand, the *direct* impact results from the use of the software during the use of the product it is integrated into. On the other hand, a software can have a number of *indirect* effects when it requires the exchange of a physically working product because of increased performance requirements of updated versions or manufacturers cease to provide updated versions of software for older hardware generations, leading to effects such as increased vulnerability to security problems.

Next we discuss the different types of impact and give an overview of methods to reduce the environmental effect and discuss the potential related benefits and drawbacks of each method.

Direct Effects: Sustainability During Use

Improving sustainability during the use of a device by reducing its energy has effects directly noticeable by its users. Especially for battery-operated portable devices, such as laptop computers and mobile phones, a reduced energy consumption during runtime will result in an extended runtime on battery power.

Vargas (2005) and Carroll and Heiser (2010) analyzed the distribution of energy consumption in mobile phones by categorizing the energy use into wireless network, central processing unit (CPU), RAM, graphics, display, and peripheral use. Both analyses show that the fraction of energy used by the CPU and RAM (i.e., the direct effects of executing software on the device) amounts to 25-40% of the device's overall energy consumption.

Another application area that is extremely critical in terms of energy consumption is the sensor network. These networks consist of a large number of small sensor nodes, embedded computing devices that communicate over a wireless network. The sensor nodes are commonly distributed over a large area (e.g., a field in case of monitoring agricultural parameters) and are either battery powered or rely on energy-harvesting methods (Alvarado et al., 2012). Exchanging depleted batteries or performing maintenance on malfunctioning energy-harvesting devices In these applications is infeasible because of economical as well as physical constraints. As a consequence, sensor network researchers have invented a large number of methods to improve the runtime of sensor nodes on battery or on limited harvested energy using a number of scheduling approaches. Although we do not concentrate specifically on sensor networks in this chapter, some of the methods discussed are also applicable to these environments, and we expect the topic of efficient sensor devices to gain increased relevance because of the increased interconnection of devices in the internet of things (IoT).

Runtime Energy Consumption Basics

The overall energy consumption of a device is $E = \int P \mathrm{d}t$ (i.e., low-power design is one way to reduce its energy consumption). On the hardware level, process technology and circuit design improvements during the last decades have enabled the reduction of power consumption while allowing higher processing speeds. However, this development hits a roadblock because of physical limitations that result in the lack of feasibility for scaling semiconductors according to Moore's law (Moore, 1965) in the future. Although using large-scale process technologies, modern CMOS circuits consume an amount of power approximately proportional to their operating frequency, $E \sim f$, Kim et al. (2003) predicted the increasing relevance of static power consumption because of leak currents when migrating semiconductors to smaller structure sizes. A more detailed analysis and prediction of these effects of nonideal scaling of semiconductors can be found in the International Technology Roadmap for Semiconductors (ITRS) roadmap (Kahng, 2013). For future complex circuits, the effect of leakage power will be paramount, in semiconductors with a large number of transistors it is no longer feasible to supply power to all transistors at the same time. As a consequence, only a part of such a chip will be active at any given time. This effect, termed “dark silicon” (Esmaeilzadeh et al., 2011), is expected to gain increasing significance for future highly complex circuits.

Because of these technological limitations on hardware, software methods to reduce the power consumption of devices have increasing relevance. Most these software approaches exploit a physical property or a specific hardware functionality in order to reduce the power consumption.

Analyzing the Energy Consumption of an Application

An obvious method for assessing the energy consumption of a given application is to measure the power consumption of the device executing the code (Tiwari et al., 1994; Russell and Jacome, 1998; Šimunić et al., 1999). Although the measuring approach results in a rather precise modeling of energy consumption, it is not practically applicable in a large number of cases. Especially in the area of embedded systems, *design space exploration* methods are used to find the optimal configuration (i.e., processor types and operating frequencies and details of the memory hierarchy) of a multicore platform. Building all required platforms in order to evaluate the energy consumption is obviously not feasible.

Energy models allowing for a flexible architecture have been developed for memories and processor architectures. The CACTI framework (Thoziyoor et al., 2008) predicts energy consumption for caches whereas DRAMsim2 (Rosenfeld et al., 2011) provides a cycle-accurate behavior and energy model for dynamic memories. The Wattch framework (Brooks et al., 2000) performs power estimation for processors on the architectural level without considering the underlying circuit or its layout. All these memory models are derived from specifications instead of measurements using a real physical device. As a consequence, they tend to be less precise.

In order to overcome this problem, *combined energy models* were developed; they predict the energy consumption of processors, communication channels, and memories. The first model was developed by Steinke et al. (2001) as a mixed model using measurements and predictions.

Steinke et al. (2001) described the overall energy consumption of a system as consisting of energy for CPU instruction execution and data processing as well as energy required for reading instructions and reading or writing data from and to memory:

$$E_{\text{total}} = E_{\text{cpu\_instr}} + E_{\text{cpu\_data}} + E_{\text{mem\_instr}} + E_{\text{mem\_data}}$$

Each class of energy consumption was subsequently refined as to its dependency on operand. For example, the cost for a sequence of *m* instructions is given as:

$$\begin{aligned} E_{\text{cpuInstr}} = {} & \sum \text{MinCostCPU}(\text{Opcode}) + \text{FUCost}(\text{Instr}_{i-1}, \text{Instr}_i) \\ & + \alpha_1 \times \sum w(\text{Imm}_{i,j}) + \beta_1 \times \sum h(\text{Imm}_{i-1,j}, \text{Imm}_{i,j}) \\ & + \alpha_2 \times \sum w(\text{Reg}_{i,k}) + \beta_2 \times \sum h(\text{Reg}_{i-1,k}, \text{Reg}_{i,k}) \\ & + \alpha_3 \times \sum w(\text{RegVal}_{i,k}) + \beta_3 \times \sum h(\text{RegVal}_{i-1,k}, \text{RegVal}_{i,k}) \\ & + \alpha_4 \times \sum w(\text{IAddr}_i) + \beta_4 \times \sum h(\text{IAddr}_{i-1}, \text{IAddr}_i) \end{aligned}$$

with *w* representing the number of “1” bits, *h* describing the Hamming distance between two values (i.e., the number of bits that flip when transitioning from one value to the next), FUCost the cost of switching functional units, and α and β parameters determined through experiments.

The work by Steinke et al. (2001) provides some interesting insights into CPU energy consumption that enabled the simplification of the resulting CPU energy model. One important result was that the Hamming distance between adjacently accessed addresses plays a major role, as shown in Figure 7.4. A number of factors could be ignored in the model. For example, it is not important which bits of an address are set to “1” or how many “1” bits an address on the memory address bus contains. Also, the cost of flipping a bit on the address bus is independent of the position of that bit. It is also irrelevant which data bit is set to “1”; however, the number of “1” bits on the data bus has a small effect (in the range of 3%) on the energy consumption. Finally, the cost of flipping a bit on the data bus is independent of the bit's position.

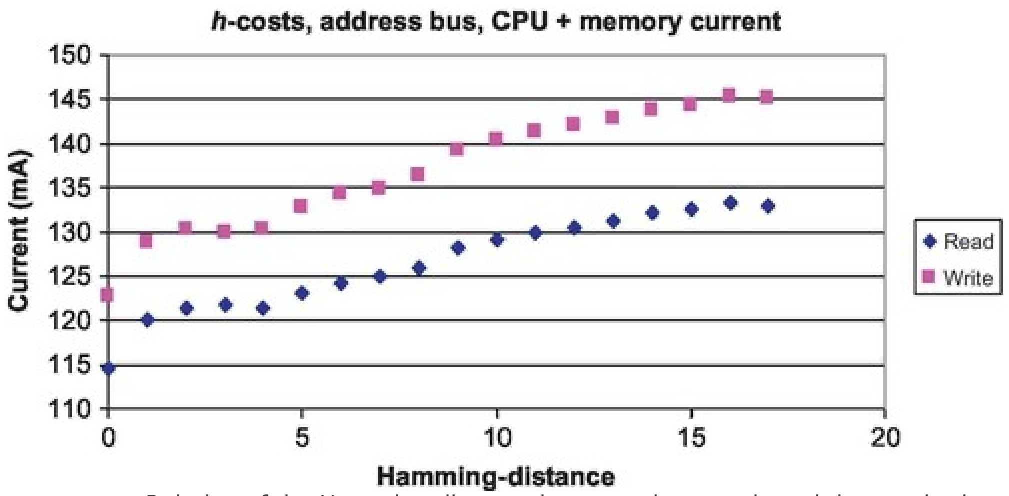


FIGURE 7.4 Relation of the Hamming distance between data words and the required current.

The CPU energy model was extended by model-based values for memories supplied by CACTI, which were validated against existing measurements of real static memory components. Using the model by Steinke et al. (2001) enabled the analysis of the energy consumption of a given sequence of machine language instructions for the ARM7TDMI processor. This model was subsequently used as the basis of research on energy-optimizing compilers.

Energy Consumption Reduction Using Physical Properties of Semiconductors

One of the commonly used approaches to conserve energy is *dynamic voltage-frequency scaling* (DVFS) (Le Sueur and Heiser, 2010). Lowering the operation frequency of a processor reduces the power consumption but will not lead to a direct reduction of energy consumption because a given program requires proportionally more time to execute. Energy savings can be achieved only because semiconductors commonly require a lower voltage supply when operating at lower frequencies. The relation of active power (ignoring leakage power) to the supply voltage V_{DD} is

$$P \sim V_{\mathrm{DD}}^2 \times f,$$

resulting in a superlinear reduction of dynamic power consumption when reducing the operating frequency and supply voltage simultaneously.

A number of additional hardware-based methods are available to reduce the power consumption. Two methods commonly used in modern semiconductors are *clock gating* (Wu et al., 2000) and *power gating* (Shi et al., 2007). Clock gating disconnects a circuit or a part of it from its driving clock(s). Because the dynamic power consumption of CMOS circuits is dependent on the operating frequency, this can be reduced to zero using clock gating. However, because of effects such as leakage currents, static power consumption is still relevant.

The dynamic and static power consumption of a circuit can be reduced to zero using power gating, which switches of the supply voltage to a circuit or a part of it. Although power gating has the obvious advantage of significant power reduction, it requires all data stored in the power gated circuit (e.g., in flip-flops or embedded memories) to be restored after the power is reapplied. Consequentially, this takes significantly longer compared to reapplying the frequency to a clock gated circuit.

Another method to reduce the energy consumption is to exploit differences in power consumption in the memory hierarchy of a system. Small static memories, called *scratchpad memories* (SPM) (Banakar et al., 2002) or *tightly coupled memories* (TCM), are significantly more energy efficient than large dynamic memories. An example for the relation of memory size to the energy required per access is given in Figure 7.5 (Marwedel et al., 2004). An approach to reduce the memory energy can rely on the locality of reference principle (Denning, 2005). This principle describes the phenomenon that a given value or related storage locations are frequently accessed by programs. *Temporal locality* refers to the reuse of specific data within a small time frame whereas *spatial locality* refers to the use of data objects within storage locations that are close to each other. Both locality patterns can be exploited to conserve energy by placing the related data objects into scratchpad memory.

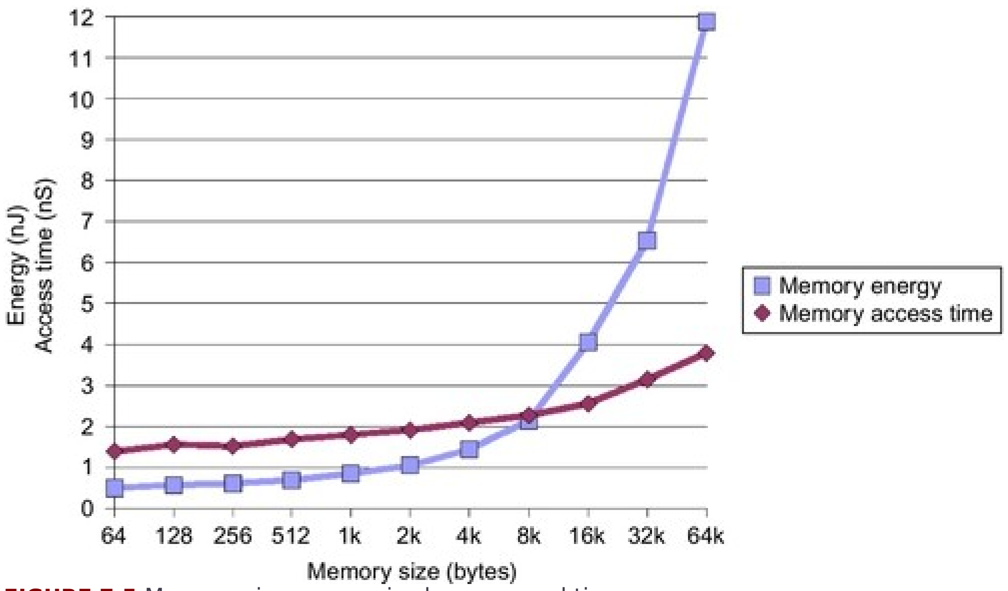


FIGURE 7.5 Memory size vs. required energy and time per access.

Optimizing the Energy Consumption of an Application: Compiler Techniques

Based on precise energy models, approaches to reduce the energy consumption can be performed either using compiler-based static analysis techniques or dynamically, using system software to adapt operational parameters of the hardware at runtime.

Early approaches to reduce the energy consumption of programs exploited standard compiler optimization techniques. Optimizations such as instruction scheduling (Parikh et al., 2000), the strength reduction of operations (e.g., replacing multiplications by a sequence of shifts and additions), and the exploitation of small bit widths of load and store operations tend to improve the performance as well as the energy consumption of code. Based on the Steinke et al.'s (2001) energy models, the first energy-optimizing compiler, *encc*, utilized an energy profiler to analyze the generated code and varied the set of applied optimizations based on feedback from the profiler (Marwedel et al., 2001). Subsequently, the topic of compiler-based energy optimization on various levels of abstraction was discussed in a number of publications. Kadayif et al. (2005) devised a method for compiler-directed high-level energy estimation and optimization. Kandemir et al. (2002) developed low-level compiler back-end optimizations for energy reduction and concluded that compiling for power/energy is different from compiling for execution cycles.

Recently, approaches that apply postcompiler optimization using genetic algorithms have been introduced for x86 systems (Schulte et al., 2014). This approach combines methods from profile-guided optimization, superoptimization, evolutionary computation, and mutational robustness. It searches for program variants that improve the nonfunctional behavior while retaining the original code semantics using characteristic workloads and predictive modeling to guide the search.

However, energy savings as a result of applying compiler back-end transformations remained rather limited in scope. One promising approach was to exploit compiler transformation and static analysis techniques to explore the locality principle and assign code and data elements to SPM.

A first approach to exploit SPMs using a nonoverlaying static allocation of code and data was presented in Steinke et al. (2002). For each memory object k, the allocation method determines its size s_k and potential gain g_k. The optimization target is then to maximize the total gain $G = \sum g_k$ under the constraint that the sum of all objects allocated to the SPM is no larger than the SPM size SSP: $\mathrm{SSP} \geq \sum s_k$. This Steinke et al. approach used a Knapsack problem solver algorithm to find an optimal partitioning of the SPM.

Optimizing the Energy Consumption of an Application: Runtime Approaches

Although static approaches to optimize energy consumption work well in environments that use only one or a small number of well-defined applications, static approaches lose their optimization potential when applied to more complex systems. In these systems, the behavior and interaction of the programs on a system may change dynamically during their execution. Thus, dynamic approaches that analyze and optimize at runtime are required.

The use of DVFS has been exploited in a large number of publications. For predictably scheduled systems, such as real-time systems, recent work was published, for example, by Lee et al. (2009), Li and Broekaert (2014), and Kim et al. (2013). Using DVFS in general-purpose systems is more complex because the requirements of tasks and their order is not known in advance. Because switching between different voltage and frequency levels requires a significant amount of energy (Muhammad et al., 2008), an incorrect prediction of future computing power requirements may result in unnecessary switching, thus reducing the possible energy savings. DVFS for general-purpose systems has been investigated, for example, in Ayoub et al. (2011), Carroll and Heiser (2014), and Curtis-Maury et al. (2008).

For memories, dynamic allocation of SPMs is more complex than its static variant. Depending on changing circumstances, code and data objects must be frequently copied into and out of the SPM, depending on the current working set of the applications. The optimization must ensure that the cost of copying objects from and to the SPM does not offset the optimization potential achieved by accessing these objects in the SPM. A first approach to implement scratchpad sharing was proposed by Verma et al. (2005) and later integrated into the scheduler of a real-time operating system by Pyka et al. (2007). Egger et al. (2008) presented a scratchpad memory manager for a preemptive multitasking system and evaluated the energy saving potential using a large number of benchmark applications. Current research focuses on optimizing systems using SPMs-introduced dynamic scratchpad memory virtualization to improve power consumption and performance of multiprocessor systems (Bathen et al., 2011).

Energy consumption can also be reduced by exploiting the heterogeneous nature of current multiprocessor systems on chip (MPSoCs). Such an MPSoC as provided by the ARM big.LITTLE architecture (Jeff, 2012) consists of a number of processor cores with different internal architectures and clock frequencies. Code can be executed using low power on a simple, slow core or, if required, accelerated on a fast, complex core. Cordes et al. (2012) presented approaches to extract task and pipeline parallelism inherent in sequential C code and perform a multicriterial optimization to find a compromise between execution time and energy consumption. This parallelization approach is unique because it is able to generate code for a heterogeneous parallel platform that is as fast as required—in contrast to as fast as possible—under given energy constraints.

Optimizing the Energy Consumption of an Application: Probabilistic Approaches

A different approach to optimize the energy consumption relies on the fact that a large number of algorithms from such diverse areas as multimedia, machine learning, and signal processing are resilient against small variations or noise in the data they operate on. Several approaches reduce the number of significant digits in floating point calculations using clock or power gating in order to save energy.

Chakrapani et al. (2006) introduced probabilistic CMOS (PCMOS), an architecture that supplies the gates of an ALU that calculate on less significant bits of a word with increasingly lower supply voltages, resulting in power savings but introducing errors into the less significant bits with a high probability because of propagation delays. The efficiency of PCMOS was evaluated on small benchmark applications as well as a large-scale scientific application for weather forecasts (Düben et al., 2014). However, Heinig et al. (2012) has shown that it is difficult to achieve using PCMOS in low-power processors using integer data types.

Sampson et al. (2011) presented EnerJ, an approximate computing framework for Java. EnerJ extends Java with type qualifiers that indicate that data in the annotated variable can be operated on using approximate computation hardware and stored in low-power memories. Like PCMOS, EnerJ also requires hardware extensions to support energy management.

Taken together, a large number of different approaches are available to dynamically optimize the energy consumption of systems at compile time as well as runtime. However, combining these approaches is not straightforward. An optimization that, considered alone provides an improvement in energy consumption, can be canceled out by another optimization operating on different hardware components. An overall framework for systemwide energy optimization that integrates a large number of the existing approaches and can perform multicriterial optimizations for a complete system is still missing and an interesting target for future research.

Indirect Effects: Sustainability vs. Production

In contrast to the direct effects of executing software, which can be measured or modeled as described previously, the indirect effects of software are harder to assess because they are densely intertwined with the life cycle of a product.

Based on their physical properties, current computing devices, such as PCs, mobile phones, and printers, can have a lifetime of ten years or more. However, most devices are used for significantly shorter periods of time. A study by Entner (2011) analyzed the frequency of handset replacement cycles. Mobile phones were on average replaced every 18.7 months in the United States, the country in the study with the highest per capita income. In contrast, in India, a handset was used for 322.1 months on average. European countries averaged between 24.5 (United Kingdom) and 53.3 (Italy) months in this study.

Obviously, except for corner cases such as accidents or stolen phones, there is no physical requirement to buy a new handset every 18 months in the United States. However, in a mostly saturated market with an estimated smartphone market penetration of more than 55% in 2014 (Deloitte, 2013), manufacturers actively search for ways to sell the next generation of handsets. The situation is similar for many electronic devices, such as computers and printers. Originally, this approach was associated with the policy of US auto makers to change their models every year to create an incentive to buy new cars. Bulow (1986) analyzed this so-called planned obsolescence, the production of goods with uneconomically short useful lives so that customers must make repeat purchases.

Thus, the question is what role software plays in the context of planned obsolescence. First, software can be used to actively *enforce* (rather than just encourage) the obsolescence of a piece of hardware. The most prominent examples are printers. Many low-cost inkjet printers employ a chip in the ink cartridges that not only ensures that only original cartridges can be used but also implements a counter that prevents them from being used after a certain threshold (number of pages, time, etc.), even though the cartridge may still contain usable ink or could be refilled. Although this is an example of software used to actually *decrease* the sustainability of printers, there is also software providing countermeasures by, for example, resetting the chip inside the cartridge. Although this software can definitely help to improve the ecological footprint—as well as the economy of operating the printer—for obvious reasons, we will not link to related resources here.

The ever-increasing demands of upcoming generations of system and application software can be interpreted as a more subtle way of planned obsolescence. Previous generation hardware often does not work or does not work well with newer software releases because they are written with respect to the capabilities of the currently available hardware. This is often used as an excuse not to provide system software upgrades after a short amount of time, especially in the Android mobile phone market. Current data on this is hard to find unless one counts the large number of complaints from users on diverse online forums. A study from 2011 (theunderstatement.com, 2011) analyzed the currentness of the Android system version for 18 different handsets on the US market at that time:

• 7 of the 18 Android phones never ran a current version of the OS.

• 12 of 18 ran a current version of the OS for only a matter of weeks or less.

• 10 of 18 were at least two major versions behind well within their two year contract period.

• 11 of 18 stopped getting any support updates less than a year after release.

• 13 of 18 stopped getting any support updates before they even stopped selling the device or very shortly thereafter.

• 15 of 18 do not run Gingerbread, which shipped in December 2010.

• When Ice Cream Sandwich comes out, every device on it will be another major version behind.

• At least 16 of 18 handsets will almost certainly never get Ice Cream Sandwich.

However, the largest problem forcing upgrades in this situation is not the fact that the devices are not able to run newer versions of apps—in fact, many owners of a phone use only a small number of built-in apps[1] (Pew Research, 2011). A much larger problem is that no security fixes are provided for older phones. With the increasing amount of malware for the Android platform (Castillo, 2011), this leaves the user in a state of fear and uncertainty and possibly more willing to upgrade a perfectly working device early.

There are a number of similar cases, for example, in the PC industry. Although newer versions of operating systems, such as Windows 7 and 8, could in theory run on older hardware, in practice a lack of drivers for older components, such as graphics cards or network controllers, prevents the installation of these newer system versions.[2]

How software can be of help here depends on the expectations and requirements of the user and often also on the user's technical abilities.

However, in general, the use of an open-source operating system helps to extend the lifetime of an older piece of equipment because support for a hardware platform is driven by developers' interest rather than economical requirements of the manufacturer. Although in the PC world, an old Windows installation could be replaced by, for example, an installation of Linux or FreeBSD, this solution will work for only a small number of users because of compatibility and usability concerns. For Android-based systems, however, compatible upgrades of the original mobile phone software are possible because of the open-source nature of the underlying system.[3] Systems such as CyanogenMod (Powers, 2011) are developed by a large number of volunteers and are made freely available for a number of older as well as current handsets, providing current software versions and bug fixed for devices without manufacturer support. However, these approaches also have their limits because some hardware components of phones rely on closed-source drivers, which are not compatible with more recent versions of the Linux kernel used in newer Android versions. Nevertheless, even in this area, some researchers provide software tools to enable the reverse engineering of binary drivers in order to enable compatibility (Chipounov and Candea, 2010).

As a consequence, the sustainability of devices could be improved if users were able to use them for a longer period of time, as also stated by Marwedel and Engel (2010). On the software side, the indirect effects of sustainability with respect to production could be mitigated when devices were provided with updates for a reasonable time after their release. Because there are no obvious economic advantages to this, support for this from manufacturers cannot be expected. Nevertheless, open-source projects such as CyanogenMod could help to improve the situation, at least for technically proficient users.

Conclusions and Outlook

This overview has shown that software has a large impact on the sustainability of electronic devices from tiny embedded sensor nodes up to PCs and servers. The major contributors to CO_2 emissions are the manufacturing of a device and its operation whereas factors such as transport and recycling are comparatively negligible.

On the one hand, the reduction of energy consumption is possible for direct causes by applying optimization methods at design and runtime of a system. These optimization approaches are based on energy measurements and models and are rather closely related to the hardware of a system and its operation modes. Although optimization approaches for single criteria have been intensively researched, an overall systemwide optimization framework that considers all parts of a system and the respective trade-offs between its components with regard to energy use is still not available.

On the other hand, a significant amount of energy can also be saved when viewing the life cycle of not only one product but also a series of products replacing each other. Because the production of a device can cause up to 50% of the overall CO_2 emissions, a device generated during its life cycle enabling its longer useful life, reduces or delays the requirement to produce a new one. This is often blocked by the manufacturers' desire to sell new devices to saturated markets by introducing planned or even forced obsolescency. Software plays an important role because limiting the availability of software upgrades to currently sold devices creates incentives to upgrade.

Some of the problems with planned obsolescency could be cured or mitigated by the use of open-source software. However, this is in no way a panacea because closed specifications of hardware devices, locked or encrypted bootloaders, or hardware fuses can provide manufacturers another way to enforce obsolescence.

The question of where we are heading remains. The limited availability of materials essential for modern electronic devices, such as raw earth elements and the ever-increasing cost of energy, will enforce sustainable production and operation of devices. Software will probably play an ever-increasing role in this because semiconductor fabrication technology heads toward the end of profitable scaling according to Moore's law.

One effect that can have interesting effects on the sustainability of software is the trend to connect mobile and stationary systems—as well as embedded systems in the IoT—to the cloud. This would enable the outsourcing of computing capacity to virtualized server farms, which can be operated more efficiently, even using a large share of renewable energy. The end user device itself would then be primarily relegated to being a sort of intelligent terminal. Some developments, such as simple laptops running Google's ChromeOS, which is essentially an execution platform for Web browser-based applications, may be pioneers of this new era. However, also this must be taken with a grain of salt. In addition to the obvious privacy concerns, the pervasive use of such devices will significantly increase the energy consumption of the mobile and wired Internet infrastructure, in turn requiring research into network energy optimization as discussed in a different chapter of this book.

References

Alvarado U, Juanicorena A, Adin I, Sedano B, Gutiérrez I, de No J. Energy harvesting technologies for low-power electronics. *Trans. Emerg. Telecommun. Technol.* 2012;23(8):728–741.

Ayoub RZ, Ogras U, Gorbatov E, Jin Y, Kam T, Diefenbaugh P, Rosing T. *OS-level power minimization under tight performance constraints in* *general purpose systems.* In: Proceedings of the 17th IEEE/ACM International Symposium on Low-Power Electronics and Design (ISLPED'11); Piscataway, NJ, USA: IEEE Press; 2011:321–326.

Banakar R, Steinke S, Lee B, Balakrishnan M, Marwedel P. *Scratchpad memory: a design alternative for Cache on-chip memory in embedded systems.* In: Proceedings of the Tenth International Symposium on Hardware/Software Codesign; New York, NY, USA: ACM; 2002:73–78.

Bathen LAD, Dutt ND, Shin D, Lim S. *SPMVisor: dynamic scratchpad memory virtualization for secure, low power, and high performance distributed on-chip memories.* In: Proceedings of the Seventh IEEE/ACM/IFIP International Conference on Hardware/software Codesign and System Synthesis (CODES + ISSS '11); New York, NY, USA: ACM; 2011:79–88. doi:10.1145/2039370.2039386.

Bozzelli, P., Gu, Q., Lago, P., 2013. A systematic literature review on green software metrics. Technical report, VU Amsterdam.

Brooks D, Tiwari V, Martonosi M. Wattch: a framework for architectural-level power analysis and optimizations. *SIGARCH Comput. Archit. News.* 2000;28(2):83–94. doi:10.1145/342001.339657.

Bulow J. An economic theory of planned obsolescence. *Q. J. Econ.* 1986;101(4):729–749. doi:10.2307/1884176.

Carroll A, Heiser G. *An analysis of power consumption in a smartphone.* In: Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference (USENIXATC'10); Berkeley, CA, USA: USENIX Association; 2010 p. 21ff.

Carroll A, Heiser G. Mobile multicores: use them or waste them. *SIGOPS Oper. Syst. Rev.* 2014;48(1):44–48. doi:10.1145/2626401.2626411.

Castillo, C.A., 2011. Android Malware: Past, Present, and Future, McAfee White Paper. http://www.mcafee.com/us/resources/white-papers/wp-android-malware-past-present-future.pdf.

Chakrapani LN, Akgul B, Cheemalavagu S, Korkmaz P, Palem KV, Seshasayee B. *Ultra efficient embedded SOC architectures based on probabilistic CMOS (PCMOS) technology.* In: Design Automation and Test in Europe Conference (DATE); 2006.

Chipounov V, Candea G. *Reverse engineering of binary device drivers with RevNIC.* In: Proceeding of the 5th ACM European Conference on Computer Systems, Paris, France, April; 2010.

Cordes D, Engel M, Marwedel P, Neugebauer O. *Automatic extraction of multi-objective aware pipeline parallelism using genetic algorithms.* In: Proceedings of CODES + ISSS; 2012:73–82.

Curtis-Maury M, Shah A, Blagojevic F, Nikolopoulos DS, de Supinski BR, Schulz M. *Prediction models for multi-dimensional power-performance optimization on many cores.* In: Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques (PACT '08); New York, NY, USA: ACM; 2008:250–259. doi:10.1145/1454115.1454151.

Deloitte, 2013. The state of the global mobile consumer, 2013—Divergence deepens.

Deng L, Babbitt CW, Williams ED. Economic-balance hybrid LCA extended with uncertainty analysis: case study of a laptop computer. *J. Clean. Prod.* 2011;19(11):1198–1206.

Denning PJ. The locality principle. *Commun. ACM.* 2005;48(7):19–24.

Dick M, Naumann S, Kuhn N. *A model and selected instances of green and sustainable software.* In: HCC'10; 2010:248–259.

Düben PD, Joven J, Lingamneni A, McNamara H, De Micheli G, Palem KV, Palmer TN. On the use of inexact, pruned hardware in atmospheric modelling. *Philos. Trans. R. Soc. A Math. Phys. Eng. Sci.* 2014;372(2018):20130276.

Egger B, Lee J, Shin H. *Scratchpad memory management in a multitasking environment.* In: Proceedings of the 7th ACM International Conference on Embedded Software (EMSOFT), Atlanta, USA, December; 2008:265–274.

Entner R. *International Comparisons: The Handset Replacement Cycle.* Massachusetts: Recon Analytics; 2011.

Esmaeilzadeh H, Blem E, St. Amant R, Sankaralingam K, Burger D. *Dark silicon and the end of multicore scaling.* In: Proceedings of the 38th Annual International Symposium on Computer Architecture (ISCA '11); New York, NY, USA: ACM; 2011:365–376. doi:10.1145/2000064.2000108.

Fujitsu. *Life Cycle Assessment and Product Carbon Footprint—Fujitsu ESPRIMO E9900 Desktop PC.* http://fujitsu.fleishmaneurope.de/?attachment_id=2148. 2010.

Heinig A, Mooney VJ, Schmoll F, Marwedel P, Palem KV, Engel M. *Classification-based improvement of application robustness and quality* *of service in probabilistic computer systems.* In: Proceedings of ARCS 2012, Munich, Germany; 2012.

Jeff B. *Big.LITTLE system architecture from ARM: saving power through heterogeneous multiprocessing and task context migration.* In: Proceedings of DAC; New York, NY, USA: ACM; 2012:1143–1146.

Kadayif I, Kandemir M, Chen G, Vijaykrishnan N, Irwin MJ, Sivasubramaniam A. Compiler-directed high-level energy estimation and optimization. *ACM Trans. Embed. Comput. Syst.* 2005;4(4):819–850. doi:10.1145/1113830.1113835.

Kahng AB. *The ITRS design technology and system drivers roadmap: process and status.* In: Proceedings of the 50th Annual Design Automation Conference (DAC '13); New York, NY, USA: ACM; 2013:doi:10.1145/2463209.2488776 Article 34, 6 pages.

Kandemir M, Vijaykrishnan N, Irwin MJ. Compiler optimizations for low power systems. In: Graybill R, Melhem R, eds. *Power Aware Computing.* Norwell, MA, USA: Kluwer Academic Publishers; 2002:191–210.

Kim NS, Austin T, Baauw D, Mudge T, Flautner K, Hu JS, Irwin MJ, Kandemir M, Narayanan V. Leakage current: Moore's law meets static power. *IEEE Comput.* 2003;36(12):68–75. doi:10.1109/MC.2003.1250885.

Kim SI, Kim HT, Kang GS, Kim J-K. *Using DVFS and task scheduling algorithms for a hard real-time heterogeneous multicore processor environment.* In: Proceedings of the 2013 Workshop on Energy Efficient High Performance Parallel and Distributed Computing (EEHPDC '13); New York, NY, USA: ACM; 2013:23–30. doi:10.1145/2480347.2480350.

Le Sueur E, Heiser G. *Dynamic voltage and frequency scaling: the laws of diminishing returns.* In: Proceedings of the 2010 International Conference on Power Aware Computing and Systems (HotPower'10); Berkeley, CA, USA: USENIX Association; 2010:1–8.

Lee WY, Ko YW, Lee H, Kim H. *Energy-efficient scheduling of a real-time task on DVFS-enabled multi-cores.* In: Proceedings of the 2009 International Conference on Hybrid Information Technology (ICHIT '09); New York, NY, USA: ACM; 2009:273–277. doi:10.1145/1644993.1645046.

Li S, Broekaert F. Low-power scheduling with DVFS for common RTOS on multicore platforms. *SIGBED Rev.* 2014;11(1):32–37. doi:10.1145/2597457.2597461.

Marwedel P, Engel M. *Plea for a holistic analysis of the relationship between information technology and carbon-dioxide emissions.* In: Workshop on Energy-aware Systems and Methods (GI-ITG), Hanover/Germany; 2010.

Marwedel P, Steinke S, Wehmeyer L. *Compilation techniques for energy-, code-size-, and run-time-efficient embedded software.* In: Int. Workshop on Advanced Compiler Techniques for High Performance and Embedded Processors, Bucharest, Hungary; 2001.

Marwedel P, Wehmeyer L, Verma M, Steinke S, Helmig U. *Fast, predictable and low energy memory references through architecture-aware compilation.* In: ASP-DAC 2004; 2004:4–11.

Moore GE. Cramming more components onto integrated circuits. *Electron. Mag.* 1965;38(8):114–117. doi:10.1109/jproc.1998.658762.

Muhammad F, Muller F, Auguin M. *Precognitive DVFS: minimizing switching points to further reduce the energy consumption.* In: Proc. 14th IEEE Real-Time and Embedded Technology and Applications Symposium, WIP Session, St. Louis, MO, USA; 2008.

Parikh A, Kandemir M, Vijaykrishnan N, Irwin MJ. *Instruction scheduling based on energy and performance constraints.* In: 2000 Proceedings. IEEE Computer Society Workshop on VLSI. IEEE Comput. Soc.; 2000:37–42. doi:10.1109/IWV.2000.844527.

Powers S. CyanogenMod 7.0—gingerbread in the house. *Linux J.* 2011;2011(206).

Purcell, K., 2011. Half of adult cell phone owners have apps on their phones. Pew research report. http://pewinternet.org/Reports/2011/Apps-update.aspx.

Pyka R, Faßbach Ch., Verma M, Falk H, Marwedel P. *Operating system integrated energy aware scratchpad allocation strategies for multiprocess applications.* In: Proceedings of 10th International Workshop on Software & Compilers for Embedded Systems (SCOPES), Nice, France; 2007:41–50.

Rosenfeld P, Cooper-Balis E, Jacob B. DRAMSim2: a cycle accurate memory system simulator. *IEEE Comput. Archit. Lett.* 2011;10(1):16–19. doi:10.1109/L-CA.2011.4.

Russell JT, Jacome MF. *Software power estimation and optimization for high performance, 32-bit embedded processors.* In: ICCD 1998; 1998:328–333.

Sampson A, Dietl W, Fortuna E, Gnanapragasam D, Ceze L, Grossman D. *EnerJ: approximate data types for safe and general low-power computation.* In: PLDI'11; 2011.

Schulte E, Dorn J, Harding S, Forrest S, Weimer W. Post-compiler software optimization for reducing energy. *SIGPLAN Not.* 2014;49(4):639–652. doi:10.1145/2644865.2541980.

Shi, K., Lin, Z., Jiang, Y.-M., 2007. Practical Power Network Synthesis for Power-Gating Designs. EETimes Design How-To, May 2007.

Šimunić et al., 1999 Šimunić T, Benini L, De Micheli G. *Cycle-accurate simulation of energy consumption in embedded systems.* In: Proceedings of the 36th Annual ACM/IEEE Design Automation Conference; 1999:867–872.

Steinke S, Knauer M, Wehmeyer L, Marwedel P. *An accurate and fine grain instruction-level energy model supporting software optimizations.* In: PATMOS, Yverdon, Switzerland; 2001.

Steinke S, Wehmeyer L, Lee B, Marwedel P. *Assigning program and data objects to scratchpad for energy reduction.* In: Proceedings of the Conference on Design, Automation and Test in Europe (DATE), Paris, France, March; 2002:409–415.

Tate K. *Sustainable Software Development: An Agile Perspective.* Boston, MA, USA: Addison-Wesley Professional; 2005.

theunderstatement.com. *Android Orphans: Visualizing a Sad History of Support.* 2011. http://theunderstatement.com/post/11982112928/android-orphans-visualizing-a-sad-history-of-support.

Thoziyoor S, Ahn JH, Monchiero M, Brockman JB, Jouppi NP. A comprehensive memory modeling tool and its application to the design and analysis of future memory hierarchies. *SIGARCH Comput. Archit. News.* 2008;36(3):51–62.

Tiwari V, Malik S, Wolfe A. Power analysis of embedded software: a first step towards software power minimization. *IEEE Trans. Very Large Scale Integr. VLSI Syst.* 1994;2(4):437–445.

Vargas O. Minimum power consumption in mobile-phone memory subsystems. *Portable Des.* 2005;11(9):24–25 Infineon Technologies.

Verma M, Petzold K, Wehmeyer L, Falk H, Marwedel P. *Scratchpad sharing strategies for multiprocess embedded systems: a first approach.* In:

Proceedings of IEEE 3rd Workshop on Embedded System for Real-Time Multimedia (ESTIMedia), Jersey City, USA, September; 2005:115–200.

Weiss RF, Mühle J, Salameh PK, Harth CM. Nitrogen trifluoride in the global atmosphere. *Geophys. Res. Lett.* 2008;35:L20821. doi:10.1029/2008GL035913.

Wu Q, Pedram M, Wu X. Clock-gating and its application to low power design of sequential circuits. *IEEE Trans. Circuits Syst. I: Fundam. Theory Appl.* 2000;47(3):415–420. doi:10.1109/81.841927.

---

[1] The study from November 2011 found out that roughly 80% of Android phone owners do not use more than ten apps on a regular basis whereas one-third of the users use only three to five apps regularly.

[2] In the case of Windows, this effect interestingly also works in the other direction—older versions of Windows (2000, XP) cannot be installed on recent hardware because of a lack of drivers.

[3] Android is based on a Linux kernel and a number of additional open-source components.

Capacity Planning for IT

Chapter 1. Goals, Issues, and Processes in Capacity Planning

THIS CHAPTER IS DESIGNED TO HELP YOU ASSEMBLE AND USE THE WEALTH OF TOOLS AND TECHNIQUES presented in the following chapters. If you do not grasp the concepts introduced in this chapter, reading the remainder of this book will be like setting out on the open ocean without knowing how to use a compass, sextant, or GPS device—you can go around in circles forever.
When you break them down, capacity planning and management—the steps taken to organize the resources your site needs to run properly—are, in fact, simple processes. You begin by asking the question: what performance do you need from your website?

First, define the application's overall load and capacity requirements using *specific* metrics, such as response times, consumable capacity, and peak-driven processing. Peak-driven processing is the workload experienced by your application's resources (web servers, databases, etc.) during peak usage. The process, illustrated in Figure 1-1, involves answering these questions:

1. How well is the current infrastructure working?
   Measure the characteristics of the workload for each piece of the architecture that comprises your applications—web server, database server, network, and so on—and compare them to what you came up with for your performance requirements mentioned above.
2. What do you need in the future to maintain acceptable performance?
   Predict the future based on what you know about past system performance then marry that prediction with what you can afford, and a realistic timeline. Determine what you'll need and *when* you'll need it.
3. How can you install and manage resources after you gather what you need?
   Deploy this new capacity with industry-proven tools and techniques.
4. Rinse, repeat.
   Iterate and calibrate your capacity plan over time.

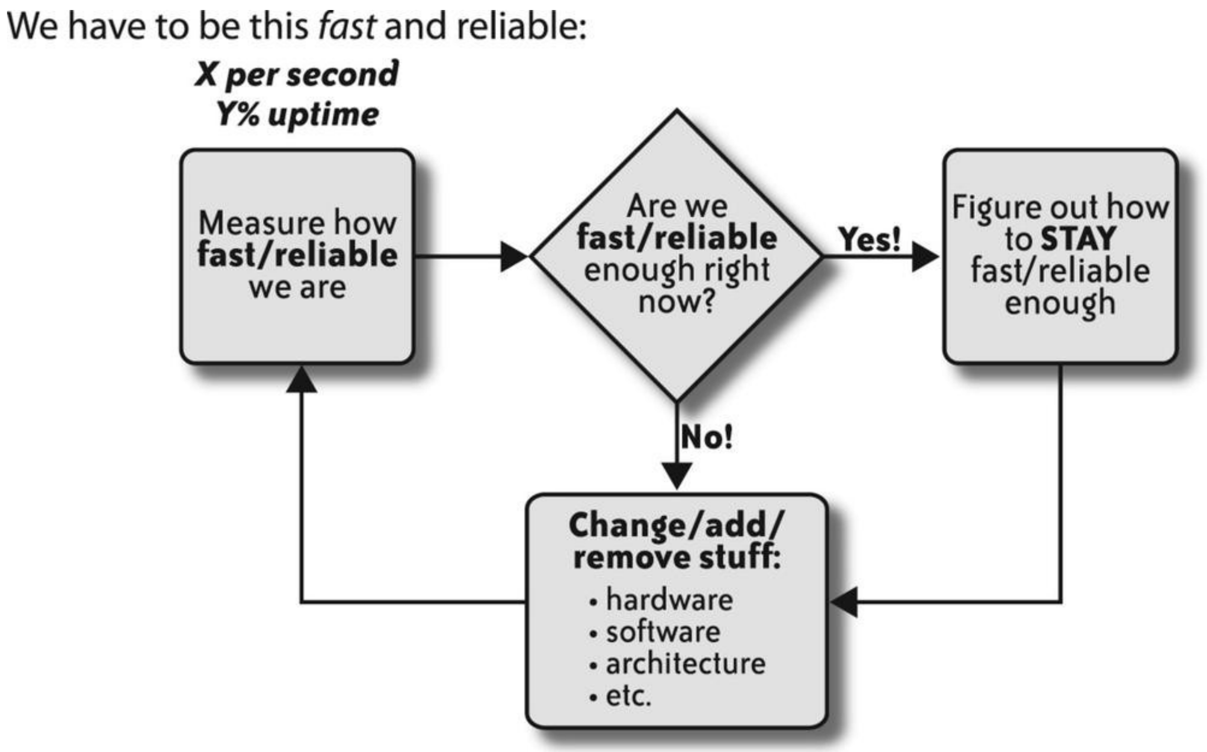

We have to be this *fast* and reliable:
X per second
Y% uptime
Measure how fast/reliable we are
Are we fast/reliable enough right now?
Yes!
Figure out how to STAY fast/reliable enough
No!
Change/add/remove stuff:
• hardware
• software
• architecture
• etc.

Figure 1-1. The process for determining the capacity you need

Your ultimate goal lies between not buying enough hardware and wasting your money on too much hardware.

Let’s suppose you’re a supermarket manager. One of your tasks is to manage the schedule of cashiers. Your challenge is picking the right number of cashiers working at any moment. Assign too few, and the checkout lines will become long, and the customers irate. Schedule too many working at once, and you’re spending more money than necessary. The trick is finding the right balance.

Now, think of the cashiers as servers, and the customers as client browsers. Be aware some cashiers might be better than others, and each day might bring a different amount of customers. Then you need to take into consideration your supermarket is getting more and more popular. A seasoned supermarket manager intuitively knows these variables exist, and attempts to strike a good balance between not frustrating the customers and not paying too many cashiers.

Welcome to the supermarket of web operations.

Quick and Dirty Math

The ideas I’ve just presented are hardly new, innovative, or complex. Engineering disciplines have always employed back-of-the-envelope calculations; the field of web operations is no different.

Because we're looking to make judgments and predictions on a quickly changing landscape, approximations will be necessary, and it's important to realize what that means in terms of limitations in the process. Being aware of when detail is needed and when it's not is crucial to forecasting budgets and cost models. Unnecessary detail means wasted time. Lacking the proper detail can be fatal.

Predicting When Your Systems Will Fail

Knowing when each piece of your infrastructure will fail (gracefully or not) is crucial to capacity planning. Capacity planning for the web, more often than one would like to admit, looks like the approach shown in Figure 1-2.

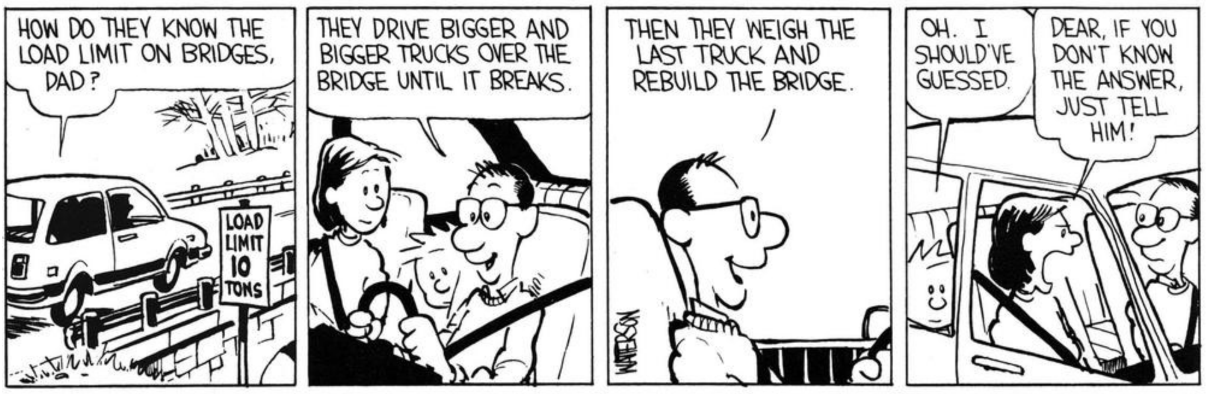

HOW DO THEY KNOW THE LOAD LIMIT ON BRIDGES, DAD?
LOAD LIMIT 10 TONS
THEY DRIVE BIGGER AND BIGGER TRUCKS OVER THE BRIDGE UNTIL IT BREAKS.
THEN THEY WEIGH THE LAST TRUCK AND REBUILD THE BRIDGE.
WATTERSON
OH. I SHOULD'VE GUESSED.
DEAR, IF YOU DON'T KNOW THE ANSWER, JUST TELL HIM!

Figure 1-2. Finding failure points

Including this information as part of your calculations is mandatory, not optional. However, determining the limits of each portion of your site's backend can be tricky. An easily segmented architecture helps you find the limits of your current hardware configurations. You can then use those capacity ceilings as a basis for predicting future growth.

For example, let's assume you have a database server that responds to queries from your frontend web servers. Planning for capacity means knowing the answers to questions such as these:

- Taking into account the specific hardware configuration, how many queries per second (QPS) can the database server manage?
- How many QPS can it serve before performance degradation affects end user experience?

Adjusting for periodic spikes and subtracting some comfortable percentage of headroom (or safety factor, which we'll talk about later) will render a single number with which you can characterize that database configuration vis-à-vis the specific role. Once you find that "red line" metric, you'll know:

- The load that will cause the database to fail, which will allow you to set alert thresholds accordingly.

- What to expect from adding (or removing) similar database servers to the backend.
- When to start sizing another order of new database capacity.

We'll talk more about these last points in the coming chapters. One thing to note is the entire capacity planning process is going to be architecture-specific. This means the calculations you make to predict increasing capacity may have other constraints specific to your particular application.

For example, to spread out the load, a LAMP application might utilize a MySQL server as a master database in which all live data is written and maintained, and use a second, replicated slave database for read-only database operations. Adding more slave databases to scale the read-only traffic is generally an appropriate technique, but many large websites (including Flickr) have been forthright about their experiences with this approach, and the limits they've encountered. There is a limit to how many read-only slave databases you can add before you begin to see diminishing returns as the rate and volume of changes to data on the master database may be more than the replicated slaves can sustain, no matter how many you add. This is just one example where your architecture can have a large effect on your ability to add capacity.

Expanding database-driven web applications might take different paths in their evolution toward scalable maturity. Some may choose to federate data across many master databases. They may split up the database into their own clusters, or choose to cache data in a variety of methods to reduce load on their database layer. Yet others may take a hybrid approach, using all of these methods of scaling. This book is not intended to be an advice column on database scaling, it's meant to serve as a guide by which you can come up with your own planning and measurement process—one that is right for your environment.

Make Your System Stats Tell Stories

Server statistics paint only part of the picture of your system's health. Unless they can be tied to actual site metrics, server statistics don't mean very much in terms of characterizing your usage. And this is something you'll need to know in order to track how capacity will change over time.

For example, knowing your web servers are processing *X* requests per second is handy, but it's also good to know what those *X* requests per second actually mean in terms of your users. Maybe *X* requests per second represents *Y* number of users employing the site simultaneously.
It would be even better to know that of those *Y* simultaneous users, *A* percent are uploading photos, *B* percent are making comments on a heated forum topic, and *C* percent are poking randomly around the site while waiting for the pizza guy to arrive. Measuring those user metrics over time is a first step. Comparing and graphing the web server hits-per-

second against those user interaction metrics will ultimately yield some of the cost of providing service to the users. In the examples above, the ability to generate a comment within the application might consume more resources than simply browsing the site, but it consumes less when compared to uploading a photo. Having some idea of which features tax your capacity more than others gives you context in which to decide where you'll want to focus priority attention in your capacity planning process. These observations can also help drive any technology procurement justifications.
Quite often, the person approving expensive hardware and software requests is not the same person making the requests. Finance and business leaders must sometimes trust implicitly that their engineers are providing accurate information when they request capital for resources. Tying system statistics to business metrics helps bring the technology closer to the business units, and can help engineers understand what the growth means in terms of business success. Marrying these two metrics together can therefore help the awareness that technology costs shouldn't automatically be considered a cost center, but rather a significant driver of revenue. It also means that future capital expenditure costs have some real context, so even those non-technical folks will understand the value technology investment brings.

For example, when presenting a proposal for an order of new database hardware, you should have the systems and application metrics on hand to justify the investment. But if you had the pertinent supporting data, you could say something along the lines of "…and if we get these new database servers, we'll be able to serve our pages *X* percent faster, which means our pageviews—and corresponding ad revenues—have an opportunity to increase up to *Y* percent." Backing up your justifications in this way can also help the business development people understand what success means in terms of capacity management.

ORA: MEASURE, MEASURE, MEASURE

Engineers like graphs for good reason: they tell a story better than numbers can by themselves, and let you know exactly how your system is performing. There are some industry-tested tools and techniques used in measuring system statistics, such as CPU, memory, and disk usage. A lot of them can be reused to measure anything you need, including application-level or business metrics.

Another theme in this book is measurement, which should be considered a necessity, not an option. You have a fuel gauge on your car's dashboard for a reason. Don't make the mistake of not installing one on your systems.

We'll see more about this in Chapter 3.

Buying Stuff: Procurement Is a Process

After you've completed all your measurements, made snap judgments about usage, and sketched out future predictions, you'll need to actually buy things: bandwidth, storage appliances, servers, maybe

even *instances* of virtual servers. In each case, you'll need to explain to the people with the checkbooks why you need what you think you need, and why you need it when you think you need it. (We'll talk more about predicting the future and presenting those findings in Chapter 4.)
Procurement is a process, and should be treated as yet another part of capacity planning. Whether it's a call to a hosting provider to bring new capacity online, a request for quotes from a vendor, or a trip to your local computer store, you need to take this important segment of time into account.

Smaller companies, while usually a lot less "liquid" than their larger bretheren, can really shine in this arena. Being small often goes hand-in-hand with being nimble. So while you might not be offered the best price on equipment as the big companies who buy in massive bulk, you'll likely be able to get it faster, owing to a less cumbersome approval process.

Quite often the person you might need to persuade is the CFO, who sits across the hall from you. In the early days of Flickr, we used to be able to get quotes from a vendor and simply walk over to the founder of the company (seated 20 feet away), who could cut and send a check. The servers would arrive in about a week, and we'd rack them in the data center the day they came out of the box. Easy!

Yahoo! has a more involved cycle of vetting hardware requests that includes obtaining many levels of approval and coordinating delivery to various data centers around the world. Purchases having been made, the local site operation teams in each data center then must assemble, rack, cable, and install operating systems on each of the boxes. This all takes more time than when we were a startup. Of course, the flip side is, with such a large company we can leverage buying power. By buying in bulk, we can afford a larger amount of hardware for a better price.

In either case, the concern is the same: the procurement process should be baked into your larger planning exercise. It takes time and effort, just like all the other steps. There is more about this in Chapter 4.

Performance and Capacity: Two Different Animals

The relationship between performance tuning and capacity planning is often misunderstood. While they affect each other, they have different goals. Performance tuning optimizes your *existing* system for better performance. Capacity planning determines what your system needs and when it needs it, using your current performance as a baseline.

ORA: COMMON SENSE STEPS AND METHODS

Real-world observations are worth more than any theoretical measurement. Capacity planning—and the predictions that drive it—should come from the *empirical* observation of your site's usage, not benchmarks made in artificial environments. Benchmarking and performance research have value, but shouldn’t be used as the sole indicators of capacity.

Let’s face it: tuning is fun, and it’s addictive. But after you spend some time tweaking values, testing, and tweaking some more, it can become a endless hole, sucking away time and energy for little or no gain. There are those rare and beautiful times when you stumble upon some obvious and simple parameter that can make everything faster—you find the one MySQL configuration parameter that doubles the cache size, or realize after some testing that those TCP window sizes set in the kernel can really make a difference. Great! But as illustrated in Figure 1-3, for each of those rare gems you discover, the amount of obvious optimizations you find thereafter dwindles pretty rapidly.

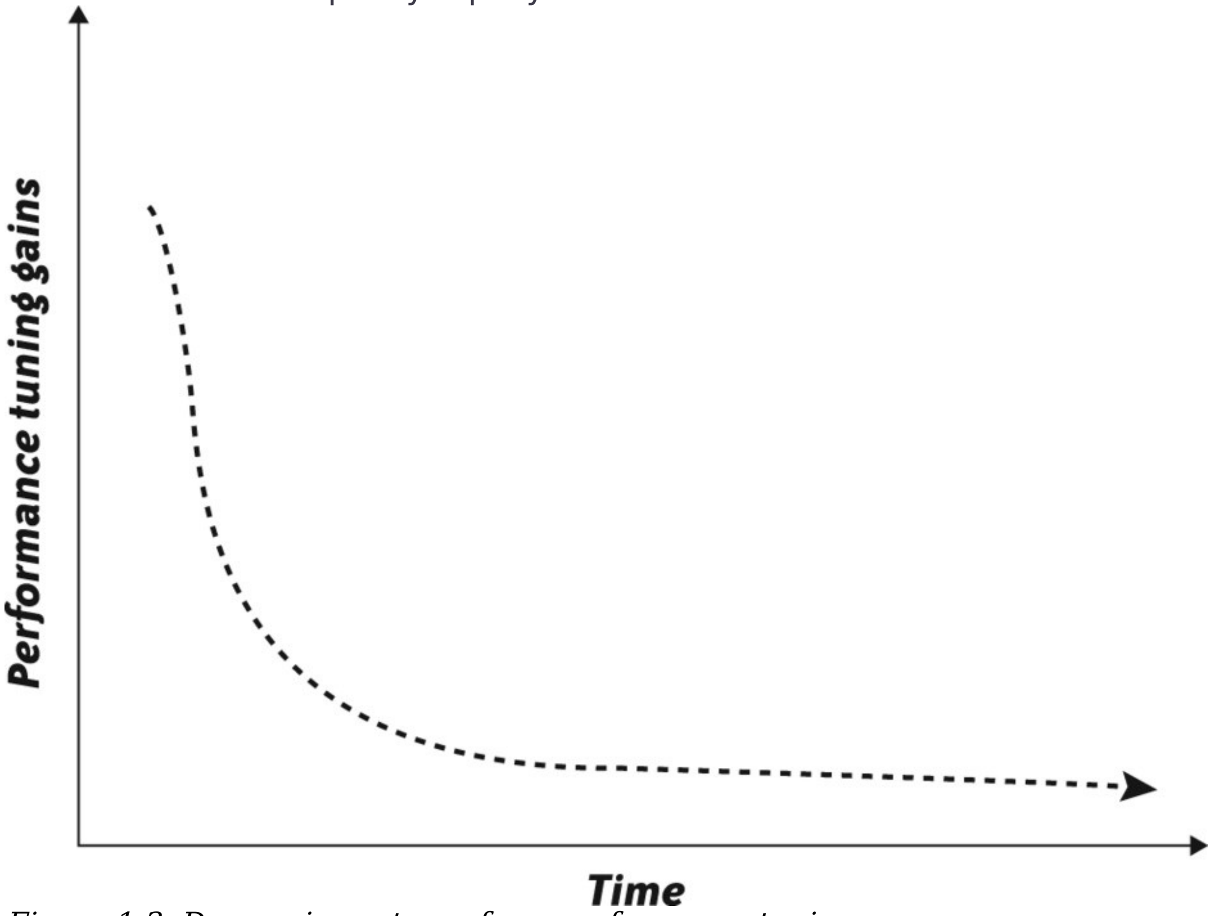


Figure 1-3. Decreasing returns from performance tuning

Capacity planning must happen *without* regard to what you might optimize. The first real step in the process is to accept the system’s *current* performance, in order to estimate what you’ll need in the future. If at some point down the road you discover some tweak that brings about more resources, that’s a bonus.

Here’s a quick example of the difference between performance and capacity. Suppose there is a butcher in San Francisco who prepares the most delectable bacon in the state of California. Let’s assume the butcher shop has an arrangement with a store in San Jose to sell their great bacon there. Every day, the butcher needs to transport the bacon from San Francisco to San Jose using some number of trucks—and the bacon has to get there within an hour. The butcher needs to determine what type of trucks, and how many of them he’ll need to get the bacon to San Jose. The demand for the bacon in San Jose is increasing with time. It’s hard having the best bacon in the state, but it’s a good problem to have.

The butcher has three trucks that suffice for the moment. But he knows he might be doubling the amount of bacon he’ll need to transport over the next couple of months. At this point, he can either:

- Make the trucks go faster
- Get more trucks

You’re probably seeing the point here. While the butcher might squeeze some extra horsepower out of the trucks by having them tuned up—or by convincing the drivers to break the speed limit—he’s not going to achieve the same efficiency gain that would come from simply purchasing more trucks. He has no choice but to accept the performance of each truck, and work from there.

The moral of this little story? When faced with the question of capacity, try to ignore those urges to make existing gear faster, and focus instead on the topic at hand: finding out what you need, and when.

One other note about performance tuning and capacity: there is no silver bullet formula to tell you when tuning is appropriate and when it’s not. It may be that simply buying more hardware is the correct thing to do, when weighed against engineering time spent on tuning the existing system. Striking this balance between optimization and capacity deployment is a challenge and will differ from environment to environment.

The Effects of Social Websites and Open APIs

As more and more websites install Web 2.0 characteristics, web operations are becoming increasingly important, especially capacity management. If your site contains content generated by your users, utilization and growth isn’t completely under the control of the site’s creators—a large portion of that control is in the hands of the user community, as shown by my example in the Preface concerning the London subway bombing. This can be scary for people accustomed to building sites with very predictable growth patterns, because it means capacity is hard to predict and needs to be on the radar of all those invested—both the business and the technology staff. The challenge for development and operations staff of a social website is to stay ahead of the growing usage by collecting enough data from that upward spiral to drive informed planning for the future.

Your driving style affects your car’s mileage. A similar principle can be applied to web architectures. One of the recurring themes in this book will be how your website’s architecture can have a significant impact on how you use, consume, and manage capacity. Design has greater effect on the effective use of your capacity than any tuning and tweaking of your servers and network. Design also plays a large role in how easily and flexibly you can add or subtract capacity as the need arises.

Although software and hardware tuning, optimization, and performance tweaking are related to capacity planning, they are not the same thing. This book focuses on tuning your architecture to allow for easier capacity management. Keeping the pieces of your architecture easily divisible and segmented can help you tackle a lot of load characterization problems—problems you’ll need to solve before you can create an accurate picture of what will be required to grow, and when.

Providing web services via open APIs introduces a another ball of wax altogether, as your application’s data will be accessed by yet more applications, each with their own usage and growth patterns. It also means users have a convenient way to abuse the system, which puts more uncertainty into the capacity equation. API usage needs to be monitored to watch for emerging patterns, usage edge cases, and rogue application developers bent on crawling the entire database tree. Controls need to be in place to enforce the guidelines or *Terms of Service* (TOS), which should accompany any open API web service (more about that in Chapter 3).
In my first year of working at Flickr, we grew from 60 photo uploads per minute to 660. We expanded from consuming 200 gigabytes of disk space per day to 880, and we ballooned from serving 3,000 images a second to 8,000. And that was just in the first year.

Capacity planning can become very important, very quickly. But it’s not all that hard; all you need to do is pay a little attention to the right factors. The rest of the chapters in this book will show you how to do this. I’ll split up this process into segments:

1. Determining your goals (Chapter 2)
2. Collecting metrics and finding your limits (Chapter 3)
3. Plotting out the trends and making forecasts based on those metrics and limits (Chapter 4)
4. Deploying and managing the capacity (Chapter 5)

1. Goals, Issues, and Processes in Capacity Planning
2. Setting Goals for Capacity
3. Measurement: Units of Capacity

Chapter 2. Setting Goals for Capacity

YOU WOULDN'T BEGIN MIXING CONCRETE BEFORE YOU KNEW WHAT YOU WERE BUILDING. SIMILARLY, you shouldn't begin planning for capacity before you determine your site's requirements. Capacity planning involves a lot of assumptions related to why you need the capacity. Some of those assumptions are obvious, others are not.

For example, if you don't know that you *should* be serving your pages in less than three seconds, you're going to have a tough time determining how many servers you'll need to satisfy that requirement. More important, it will be even tougher to determine how many servers you'll need to add as your traffic grows.

Common sense, right? Yes, but it's amazing how many organizations don't take the time to assemble a rudimentary list of operational requirements. Waiting until users complain about slow responses or time-outs isn't a good strategy.

Establishing the acceptable speed or reliability of each part of your site can be a considerable undertaking, but it will pay off when you're planning for growth and need to know what standard you should maintain. This chapter shows you how to understand the different types of requirements your management and customers will force you to deal with, and how architectural design helps you with this planning.

Different Kinds of Requirements and Measurements

Now that we're talking about requirements—which might be set by others, external to your group—we can look at the different types you'll need to deal with. Your managers, your end-users, and your clients running websites with you, all have varying objectives and measure success in different ways. Ultimately, these requirements, or capacity goals, are interrelated and can be distilled into the following:

- Performance
  — External service monitoring
  — Business requirements
  — User expectations
- Capacity
  — System metrics
  — Resource ceilings

Interpreting Formal Measurements

Your site should be available not only to your colleagues performing tests on your website from a facility down the road, but also to real visitors who may be located on other continents with slow connections. Some large companies choose to have site performance (and availability) constantly monitored by services such as Keynote (http://keynote.com) or Gomez (http://gomez.com). These commercial services deploy worldwide networks of machines that constantly ping your web pages to record the return time. Servers then keep track of all these metrics and build you a handy-dandy dashboard to evaluate how your site performance and uptime appears from many locations around the world. Because Keynote and Gomez are deemed "objective" third parties, those statistics can be used to enforce or guide Service Level Agreements (SLAs) arranged with partner companies or sites (we'll talk more about SLAs later). Keynote and Gomez can be considered enterprise-level services. There are also plenty of low-cost alternatives, including PingDom (http://pingdom.com), SiteUptime (http://siteuptime.com), and Alertra (http://alertra.com).

It's important to understand exactly what these services measure, and how to interpret the numbers they generate. Since most of them are networks of *machines* rather than people, it's essential to be aware of how those web pages are being requested. Some things to consider when you're looking at service monitoring systems include:

- Are they simulating human users?
- Are they caching objects like a normal web browser would? Why or why not?
- Can you determine how much time is spent due to network transfer versus server time, both in the aggregate, and for each object?
- Can you determine whether a failure or unexpected wait time is due to geographic network issues or measurement failures?

If you believe your service monitoring systems are testing in a manner representative of your users when they visit your site, you have good reasons to trust the numbers. Also keep in mind, the metrics you use for capacity planning or site performance measurement might ultimately find their way onto an executive dashboard somewhere, viewed by a non-technical audience.

CFOs, CTOs, business development folks, and even CEOs can become addicted to qualitative assessments of operations. This can be a double-edged sword. On the one hand, you're being transparent about failures, which can help when you're attempting to justify expenditures and organizational changes to support capacity. On the other hand, you're also giving a frequently obsessive crowd more to obsess about, so when there are any anomalies in this data, you should be prepared to explain what they mean.

Service Level Agreements

So what exactly is an SLA? It's an instrument that makes business people comfortable, much like insurance. But in broader, less anxious terms, an SLA is a metric that defines how a service should operate within agreed-upon boundaries. It puts some financial muscle into the metric by establishing a schedule of credits for meeting goals, or possibly penalties if the service does not achieve them. With websites, SLAs cover mostly *availability* and *performance*.

Some SLAs guarantee a service will available for a pre-established percentage of time, such as 99.99%. What this means is that 0.01% of the time, the service can be unavailable, and it will still be within the bounds of the SLA. Other SLAs require that demand for a service stay within reasonable limits; request rate limits or storage and upload limits are typical parameters.

For example, you might find a web hosting company with something like verbiage below in its "Terms of Service" document:

Acme Hosting, Inc. will use commercially reasonable efforts to make the SuperHostingPlan available with a Monthly uptime percentage (defined below) of at least 99.9% during any monthly billing cycle. In the event Acme Hosting, Inc. does not meet this commitment, you will be eligible to receive a service credit as described below.

Monthly uptime percentage	Credit percentage
Between 99 and 99.9%	1 day credit
Less than 99%	1 week credit

Looks pretty reassuring, doesn't it? The problem is, 99.9% uptime stretched over a month isn't as great a number as one might think:

30 days = 720 hours = 43,200 minutes
99.9% of 43,200 minutes = 43,156.8 minutes
43,200 minutes – 43,156.8 minutes = 43.2 minutes

This means for 43.2 minutes every month, this service can go down without penalty. If your site generates $3,000 worth of sales every minute, you could easily calculate how much money any amount of downtime will cost you (along with the less measurable consequence of disgruntled customers). Table 2-1 shows percentages of uptime on a yearly basis.

Table 2-1. SLA percentages and acceptable downtimes

Uptime SLA	Downtime per year
90.0%	36 days, 12 hours
95.0%	18 days, 6 hours
99.0%	87 hours, 36 minutes
99.50%	43 hours, 48 minutes
99.90%	8 hours, 45 minutes, 36 seconds
99.99%	52 minutes, 33 seconds
99.999%	5 minutes, 15 seconds
99.9999%	32 seconds

The term *five-nines* is commonly heard in discussions about SLAs and availability. This refers to 99.999% availability, and it is used in marketing literature at least as much as it is in technical literature. Five-nines is usually used to indicate your site or system is deemed to be *highly* available.

These SLA availability numbers aim to provide a level of confidence in a website’s service, but also imply you can equate downtime to lost revenue. I don’t believe this is actually accurate, as the straight math will bear out. If your service is unavailable for 10 minutes and it normally produces $3,000 of revenue every minute, then you might assume your business has lost $30,000. In reality, customers might just pick up where they left off and buy what they were in the process of buying when the outage occurred. Your business might be spending extra money on the customer service side to make up for an outage that has no impact on your earnings.

The point is, while a true and accurate financial representation of an outage may be neither true nor accurate, the importance of availability should be clear.

Business Capacity Requirements

The use of *web services* is becoming more and more prevalent in today's Web 2.0 mashup-y world. While most web services offer open APIs for individual application developers to build upon, business-to-business relationships depend on them as well. Therefore, companies usually tie revenue streams to having unfettered access to an API. This could mean a business relationship relies on a certain level of availability, or performance of your API, measured in a percentage uptime (such as 99.99%) or an agreed-upon rate of API requests.

Let's assume your website provides postal codes, given various inputs to the API you've built. You might allow only one API call per minute to a regular or non-commercial user, but a shipping company might enter into a contract permitting it to call your API up to 10 times per *second*. Website capacity planning is as much about justifying capital expenditures as it is about technical issues, such as scaling, architectures, software, and hardware. Because capacity concerns can have such a large impact on *business* operations, they should be considered early in the process of development.

User Expectations

Obviously, the end goal of capacity planning is a smooth and speedy experience for your users. Several factors can affect the user's experience beside capacity. It's possible to have plenty of capacity but a slow website nonetheless. Designing fast web pages is beyond the scope of this book, but you can find a lot of great information in Steve Souders' excellent book, *High Performance Web Sites* (O'Reilly).

Even though capacity is only one part of making the end-user experience fast, that experience is still one of the real-world metrics that we'll want to measure and track in order to make forecasts.

For example, when serving static web content, you may reach an intolerable amount of latency at high volumes before any system-level metrics (CPU, disk, memory) raise a red flag. Again, this can have more to do with the construction of the web page than the capacity of the servers sending the content. But as capacity is one of the more expensive pieces to change, it warrants investigation. Perceived slowness of a web page *could* be the result of a page that is simply too heavy, and not from a lack of capacity. (This is one of the fundamentals of Souders' book.) It's a good idea to determine whether this is the case when any user-perceived slowness is analyzed. The problem can be solved by either 1) adding capacity, or, 2) changing the page weight. The first solution can sometimes involve more cost than solution two.

At Flickr, we serve tens of thousands of photos per second. Each photo server can serve a known and specific rate of images before reaching its maximum. We don't define this maximum in terms of disk I/O, or CPU, or memory, but in terms of how many images we can serve without the "time to serve" for each image exceeding the specified amount of time.

Architecture Decisions

Your architecture is the basic layout of how all of the backend pieces—both hardware and software—are joined. Its design plays a crucial role in your ability to plan and manage capacity. Designing the architecture can be a complex undertaking, but there are a couple of great books available to help you: Cal Henderson's *Building Scalable Web Sites* (O'Reilly) and Theo Schlossnagle's *Scalable Internet Architectures* (Pearson).
Your architecture affects nearly every part of performance, reliability, and management. Establishing good architecture almost always translates to easier effort when planning for capacity.

Providing Measurement Points

Both for measurements purposes as well as for rapid response to changing conditions, you want your architecture to be designed so you can easily split it into parts that perform discrete tasks. In an ideal world, each component of the backend should have a single job to do, but it could still do multiple jobs well, if needed. At the same time, its effectiveness on each job should be easy to measure.

For instance, let's look at a simple, database-driven web application just starting on its path toward world domination. To get the most bang for our buck, we have our web server and our database residing on the same hardware server. This means all the moving parts share the same hardware resources, as shown in Figure 2-1.

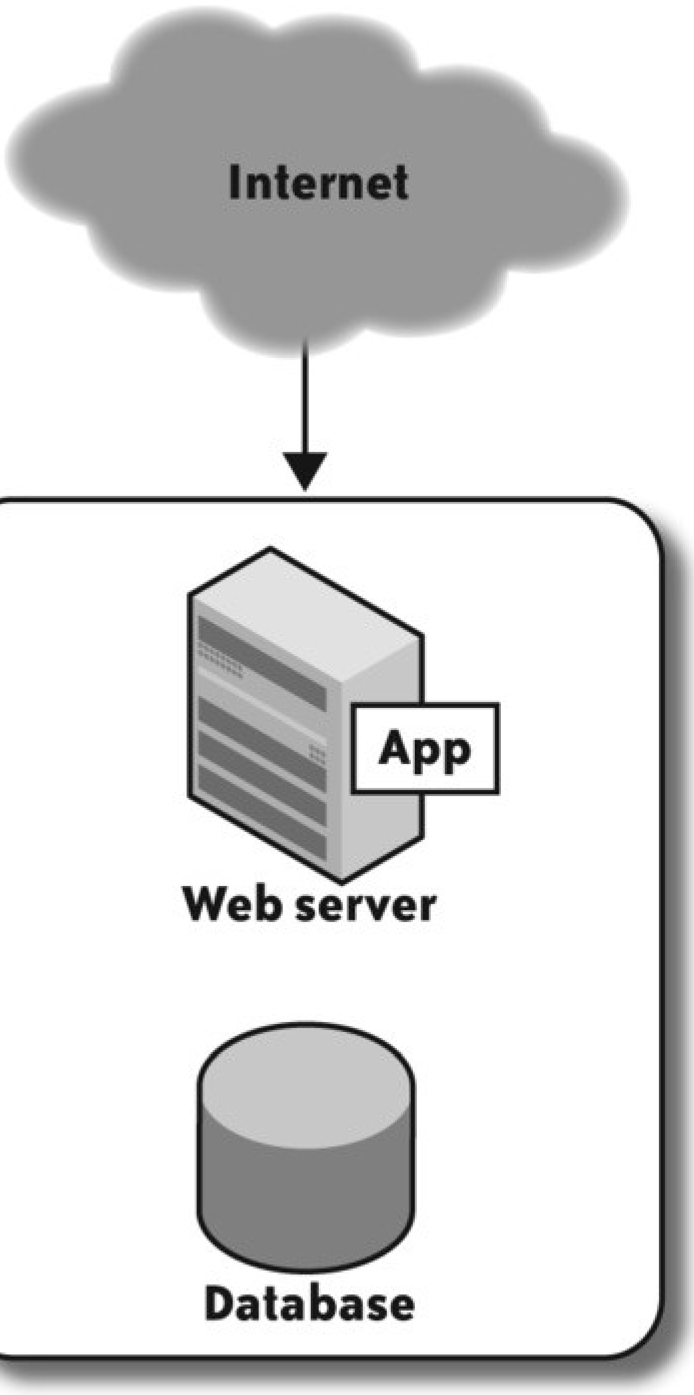


Figure 2-1. A simple, single-server web application architecture

Let’s suppose you’ve already read Chapter 3 (cheating, are we?) and you have configured measurements for both system and application-level statistics for your server. You can measure the system statistics of this server via `sar` or `rrdtool`, and maybe even application-level measurements such as web resource requests or database queries-per-second.

The difficulty with the setup in Figure 2-1 is you can’t easily distinguish which system statistics correspond with the different pieces of the architecture. Therefore, you can’t answer basic questions that are likely to arise, such as:

- Is the disk utilization the result of the web server sending out a lot of static content from the disk, or rather, the database’s queries being disk-bound?
- How much of the filesystem cache, CPU, memory, and disk utilization is being consumed by the web server, and how much is being used for the database?

With careful research, you can make some estimates about which daemon is using which resource. In the best case, the resource demands of the different daemons don’t contend with one another. For example, the web server might be bound mostly by CPU and not need much memory, whereas the database might be memory-bound without using much CPU. But even in this ideal scenario, if usage continues to grow, the resource contention will grow to warrant splitting the architecture into different hardware components (Figure 2-2). At that point, you’d really like to know how much CPU, cache, disk space, bus bandwidth, and so on, each daemon actually needs.

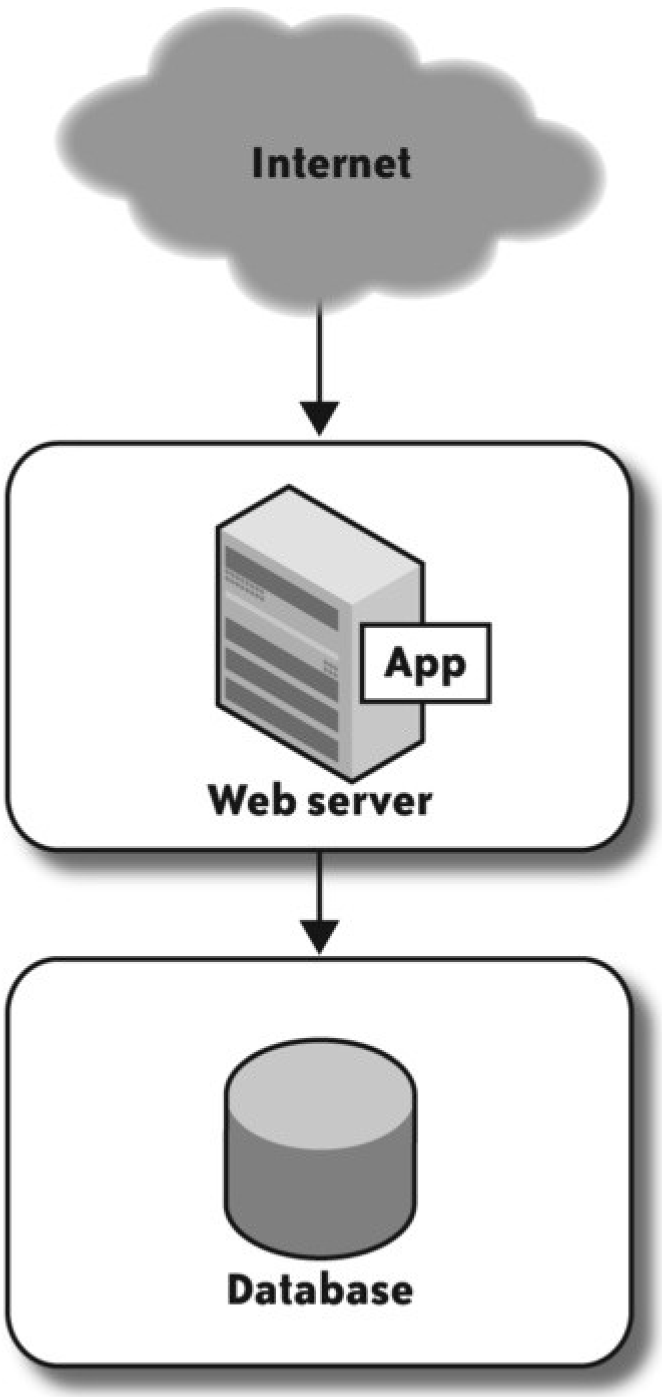


Figure 2-2. Separation of web server and database

Splitting the nodes in this fashion makes it easier to understand the capacity demands, as the resources on each server are now dedicated to each piece of the architecture. It also means you can measure each server and its resource demands more distinctly. You could come to conclusions with the single-component configuration, but with less ease and accuracy. Of course, this division of labor also produces performance gains, such as preventing frontend client-side traffic from interfering with database traffic, but let’s forget about performance for the moment.

If we’re recording system and application-level statistics, you can quantify what each unit of capacity means in terms of usage. With this new architecture, you can answer a few questions that you couldn’t before, such as:

Database server
How do increases in database queries-per-second affect the following?

- Disk utilization
- I/O Wait (percent of time the database waits due to network or disk operations)
- RAM usage
- CPU usage

Web server
How do increases in web server requests-per-second affect the following?

- Disk utilization
- I/O Wait
- RAM usage
- CPU usage

Being able to answer these questions is key to establishing how (and when) you’ll want to add more capacity to each piece.

Providing Scaling Points

Now that you have a good idea of what’s required for each piece of this simple architecture, you can get a sense for whether you’ll want different hardware configurations.

At Flickr, for the most part, our MySQL database installations happen to be disk-bound, so there’s no compelling reason to buy two quad-core CPUs for each database box. Instead, we spend money on more disk spindles and memory to help with filesystem performance and caching. We know this to be our ideal database hardware configuration—for our database. We have different configurations for our image serving machines, our web servers, and our image processing machines; all according to what in-box resources they rely on most.

The last piece we’re missing in this discussion on architecture is what drives capacity forecasting: *resource ceilings*. The questions posed earlier regarding the effects of usage on resources, point to an obvious culmination: *when will the database or web server die?*
Each server in our example possesses a finite amount of the following hardware resources:

- Disk throughput
- Disk storage
- CPU
- RAM
- Network

High loads will bump against the limits of one or more of those resources. Somewhere just below that critical level is where you'll want to determine your *ceiling* for each piece of your architecture. Your ceiling is the critical level of a particular resource (or resources) that cannot be crossed without failure. Armed with your current ceilings, you can start to assemble your capacity plan. We'll talk more about examples of ceilings in Chapter 3.

As you can see, changing architecture in simple ways can help you understand for what purposes your capacity is being used. When thinking about architecture design, keep in mind the division of labor and the "small pieces, loosely joined" theory can go a long way toward giving you clues as to how your site is being used. We'll touch more on architecture decisions throughout the book, and particularly in Chapter 3.

Hardware Decisions (Vertical, Horizontal, and Diagonal Scaling)

Choosing the right hardware for each component of your architecture can greatly affect costs. At the very least, when it comes to servers, you should have a basic idea (gleaned from measurement and usage patterns) of where you want to invest your money.

Before perusing your vendor's current pricing, be aware of what you're trying to achieve. Will this server be required to do a lot of CPU work? Will it need to perform a lot of memory work? Is it a network-bound gateway?

Today, the difference between horizontal and vertical scaling architectures are quite well known in the industry, but it bears reviewing in order to put capacity planning into context.

Being able to scale *horizontally* means having an architecture that allows for adding capacity by simply adding similarly functioning nodes to the existing infrastructure. For instance, a second web server to share the burden of website visits.

Being able to scale *vertically* is the capability of adding capacity by increasing the resources internal to a server, such as CPU, memory, disk, and network.

Since the emergence of tiered and *shared-nothing* architectures, horizontal scaling has been widely recognized for its advantages over vertical scaling as it pertains to web applications. Being able to scale horizontally means designing your application to handle various levels of database abstraction and distribution. You can find great approaches to horizontal application development techniques in the aforementioned books by Henderson and Schlossnagle.

The danger of relying *solely* on vertical scaling is, as you continue to upgrade components of a single computer, the cost rises dramatically. You also introduce the risk of a *single point of failure* (SPOF).
Horizontal scaling involves the more complex issue of increasing the potential failure points as you expand the size of the server farm. In addition, you inherently introduce some challenges surrounding any synchronization you'll need between the nodes.

Diagonal scaling (a term coined by myself) is the process of vertically scaling the horizontally scaled nodes you already have in your infrastructure. Over time, CPU power and RAM become faster, cheaper, and cooler, and disk storage becomes larger and less expensive, so it can be cost effective to keep some vertical scaling as part of your plan, but applied to horizontal nodes.
What this all boils down to is, for all of your nodes bound on CPU or RAM, you can "upgrade" to fewer servers with more CPU and RAM. For disk-bound boxes, it can also mean you may be able to replace them with fewer machines that have more disk spindles.

As an example, I'll take a recent Flickr upgrade.

Initially, we had 67 dual-CPU, 4 GB RAM, single SATA drive web servers. For the most part, our frontend layer is CPU-bound, handling requests from client browsers, making backend database calls, and taking photo uploads. These 67 machines were equipped with Intel Xeon 2.80 GHz CPUs running Apache and PHP.

When it was time to add capacity, we decided to try the new Quad Core CPU boxes. We found the dual quad core machines had roughly three times the processing power of the existing dual CPU boxes. With 8 CPU cores of Intel Xeon L5320 1.86 GHz CPUs, we were able to replace 67 existing boxes with only 18 new boxes. Figure 2-3 illustrates how much the server load average (across the entire cluster) dropped as a result.
Figure 2-3 shows the reduction in load average when the 67 machines were removed from the production pool and the 18 new boxes were allowed to take over for the same production load. This certainly makes for a very dramatic-looking graph, but load average might not be the best metric to illustrate this diagonal scaling exercise.
Figure 2-4 represents the same time period as Figure 2-3, except it details the number of apache requests-per-second when the older servers were replaced. The shades of lines on the graph represent a single server, allowing you to clearly see when the newer servers took over. Note the amount of apache requests-per-second actually went up by as much as 400 after the replacement, implying the older machines were very close to their own bottlenecks.
Let's take a look at Table 2-2 to learn what this meant in terms of resources.

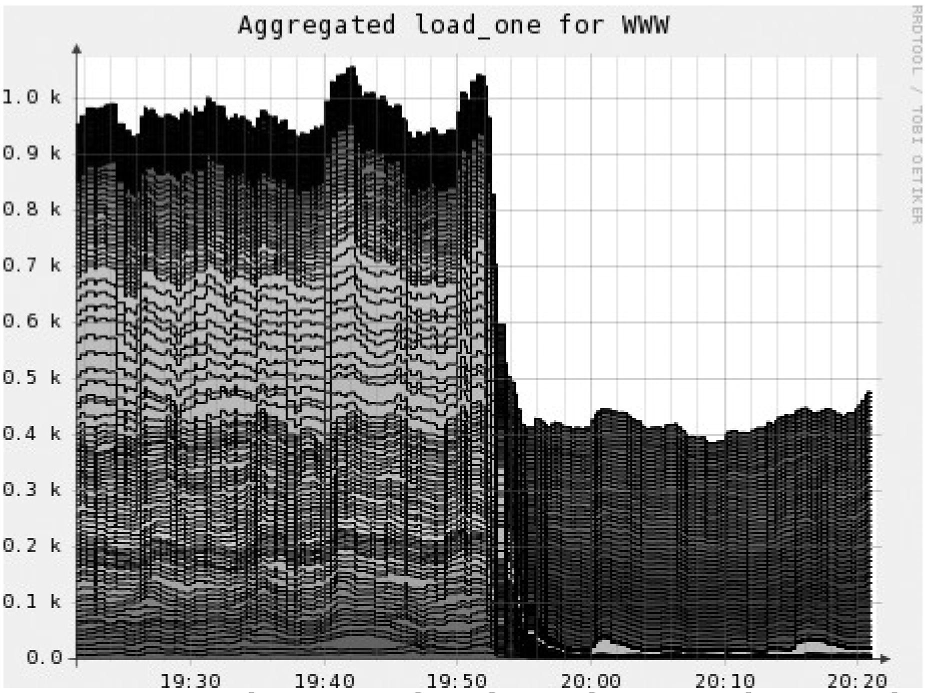


Figure 2-3. Load average drop by replacing 67 boxes with 18 higher capacity boxes

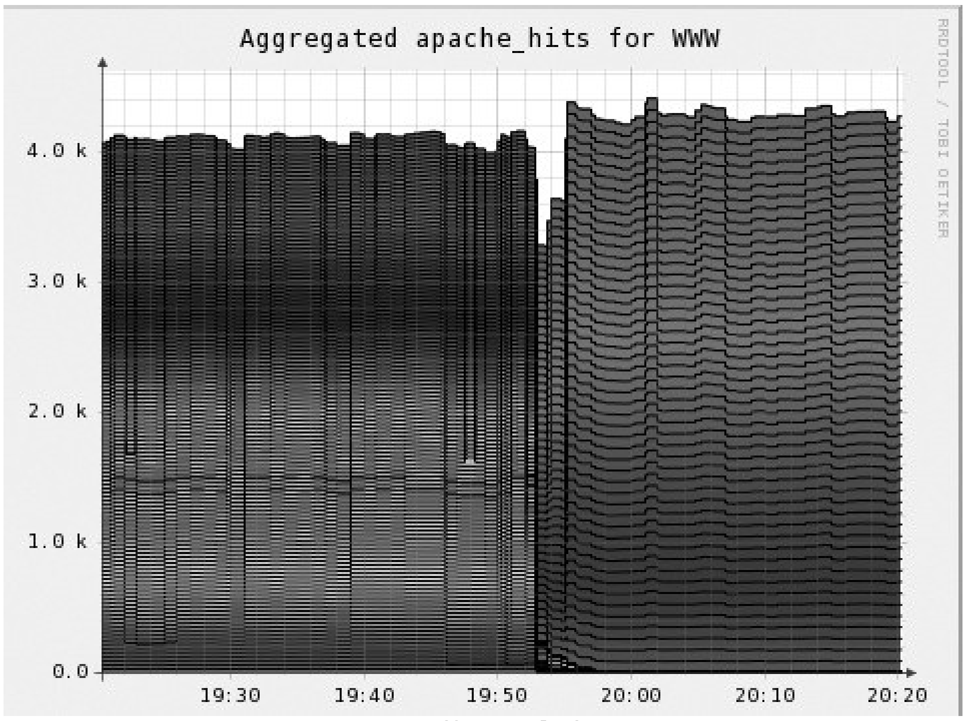


Figure 2-4. Serving more traffic with fewer servers

Table 2-2. Comparing server architectures

Servers	CPU	RAM	Disk	Power (kW) at 60% of peak usage
67	2 (2 cores)	4 GB	1 x 80 GB SATA	8.763

Servers	CPU	RAM	Disk	Power (kW) at 60% of peak usage
18	2 (8 cores)	4 GB	1 x 146 GB SATA	2.332

Based on traffic patterns, if we assume the servers are working at an average of about 60 percent of their peak, this means we're using roughly 30 percent of the electrical power we were using previously. We've also saved 49U of rack space because each server needs only 1U of space. That's more than one full, standard 42U rack emptied as a result of diagonal scaling. Not bad.

Disaster Recovery

Disaster recovery is saving business operations (along with other resources, such as data, which we won't consider in this book) after a natural or human-induced catastrophe. By catastrophe, I'm not implying the failure of a single server, but a complete outage that's usually external to the operation of the website infrastructure.
Examples of such disasters include data center power or cooling outages, as well as physical disasters, such as earthquakes. It can also include incidents, such as construction accidents or explosions that affect the power, cooling, or network connectivity relied upon by your site. Regardless of the cause, the effect is the same: you can't serve your website. Continuing to serve traffic under failure conditions is obviously an important part of web operations and architecture design. Contingency planning clearly involves capacity management. *Disaster recovery* (DR) is only one part of what is termed *Business Continuity Planning* (BCP), which is the larger logistical plan to ensure continuity of business in the face of different failure event scenarios.
In most cases, the solution is to deploy complete architectures in two (or more) separate physical locations, which means multiplying your infrastructure costs. It also means multiplying the nodes you'll need to manage, doubling all of the data replication, code, and configuration deployment, and multiplying all of your monitoring and measurement applications by the number of data centers you deploy.

Clearly, DR plans raise both economic and technical concerns. DR and BCP are large topics in and of themselves, and are beyond the scope of this book. If this topic is of particular interest to you, there are many books available dedicated specifically to this subject.

2. Setting Goals for Capacity

3. Measurement: Units of Capacity

4. Predicting Trends

4h 13m remaining

Chapter 3. Measurement: Units of Capacity

The only man who behaves sensibly is my tailor; he takes my measurements anew every time he sees me, while all the rest go on with their old measurements and expect me to fit them.
— George Bernard Shaw

IF YOU DON'T HAVE A WAY TO MEASURE YOUR CURRENT CAPACITY, YOU CAN'T CONDUCT capacity planning—you'll only be guessing. Fortunately, a seemingly endless range of tools is available for measuring computer performance and usage. I'm willing to bet that moments after the first computer program was written, another one was written to measure and record how fast the first one performed.

Most operating systems come with some basic built-in utilities that can measure various performance and consumption metrics. Most of these utilities usually provide a way to record results as well. Additional popular open source tools are easy to download and run on virtually any modern system. For capacity planning, your measurement tools should provide, at minimum, an easy way to:

- Record and store data over time
- Build custom metrics
- Compare metrics from various sources
- Import and export metrics

As long as you choose tools that can in some way satisfy this criteria, you don't need to spend much time pondering which to use. What is more important is what metrics you choose to measure, and what metrics to which you pay particular attention.

In this chapter, I'll discuss the specific statistics you'll want to measure for different purposes, and show the results in graphs to help you better interpret them. There are plenty of other sources of information on how to set up particular tools to generate the measurements; most professional system administrators already have such tools installed.

ORA: ACCEPTING THE OBSERVER EFFECT

Measuring your systems introduces yet another task your server will be asked to perform in order to function properly. Some system resources are going to be consumed for the purposes of collection and transfer of metrics. Good monitoring tools make an effort to be lightweight and not get in the way of a server's primary work, but there will always be some amount of overhead. This means simply measuring your system's resources will in some small way (hopefully, very small) affect the system's behavior, and by extension, the very measurements you end up recording. This is commonly known as the "observer effect."

My philosophy is to accept the burden on the server and the slight distortion in the data collected as a cost of doing business. Giving up some percentage of CPU, disk, memory, and network resources to provide clear and useful measurement data is a small price to pay for monitoring your system's overall health and capacity.

Aspects of Capacity Tracking Tools

This chapter is about automatically and routinely measuring server behavior over a predefined amount of time. By monitoring normal behavior over days, weeks, and months, you'll be able to see both patterns that recur regularly, and trends over time that help you predict when you need to increase capacity.

We'll also discuss deliberately increasing the load through artificial scaling using methods that closely simulate what will happen to your site in the future. This will also help you predict the need to increase capacity.

For the tasks in this chapter, you need tools that collect, store, and display (usually on a graph) metrics over time. They can be used to drive capacity predictions as well as problem resolution.

Examples of these tools include:

Cacti (http://cacti.net)

Munin
(http://munin.projects.linpro.no/)

Ganglia (http://ganglia.info)

Hyperic HQ (http://hyperic.com)

The tools don't need to be fancy. In fact, for some metrics, I still simply load them into Excel and plot them there. Appendix C contains a more comprehensive list of capacity planning tools.
It's important to start out by understanding the types of monitoring to which this chapter refers. Companies in the web operations field use the term *monitoring* to describe all sorts of operations—generating alerts concerning system availability, data collection and its analysis, real-world and artificial end user interaction measurement—the list goes on and on. Quite often this causes confusion. I suspect many commercial vendors who align on any one of those areas exploit this confusion to further their own goals, much to our detriment as end users.
This chapter is *not* concerned with system availability, the health of your servers, or notification management—the sorts of activities offered by Nagios, Zenoss, OpenNMS, and other popular network monitoring systems. Some of these tools do offer some of the features we need for our monitoring purposes, such as the ability to display and store metrics. But they exist mostly to help you recognize urgent problems and avoid imminent disasters. For the most part, they function a lot like extremely complex alarm clocks and smoke detectors.

Metric collection systems, on the other hand, act more like court reporters, who observe and record what's going on without taking any action whatsoever. As it pertains to our goals, the term monitoring refers to metric collection systems used to collect, store, and display system and application-level metrics of your infrastructure.

Fundamentals and Elements of Metric Collection Systems

Nearly every major commercial and open source metric collection system employs the same architecture. As depicted in Figure 3-1, this architecture usually consists of an *agent* that runs on each of the physical machines being monitored, and a single *server* that aggregates and displays the metrics. As the number of nodes in your infrastructure grows, you will probably have more than a single server performing aggregation, especially in the case of multiple data center operations.

The agent's job is to periodically collect data from the machine on which it's running and send a summary to the metric aggregation server. The metric aggregation server stores the metrics for each of the machines it's monitoring, which can then be displayed by various methods. Most aggregation servers use some sort of database; one specialized format known as *Round-Robin Database* (RRD) is particularly popular.

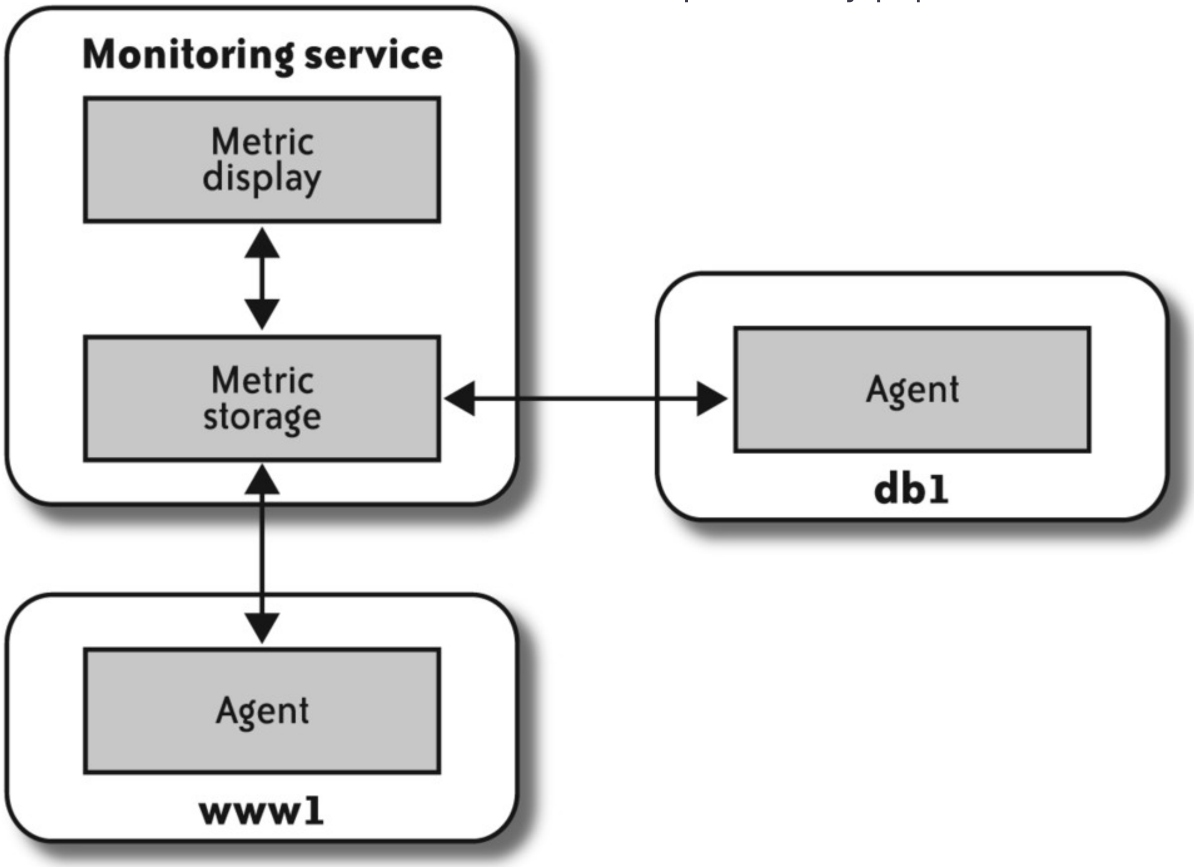


Figure 3-1. The fundamental pieces of most metric collection systems

Round-Robin Database and RRDTool

RRDTool is probably the most commonly used utility for storing system and network data—at least for those using the LAMP stack. I'm only going to offer you an overview here, but you can find a full description of it on the "about" page and in the tutorials of its RRDTool website at http://rrdtool.org.
The key characteristics of system monitoring data concern its size: there's a lot of it, and it's constantly increasing. Thus, ironically, you need to do capacity planning just for the data you're collecting for capacity planning! The Round-Robin Database (RRDTool) utility solves that by making an assumption that you're interested in fine details only for the recent past. As you move backward in the stored data, it's acceptable to lose some of the details. After some maximum time defined by the user (say, a year), you let data disappear completely. This approach sets a finite limit on how much data you're storing, with the tradeoff being the degree of detail as time moves on.

RRDTool can also be used to generate graphs from this data and show views on the various time slices for which you've recorded data. It also contains utilities to dump, restore, and manipulate RRD data, which come in handy when you drill down into some of the nitty-gritty details of capacity measurement. The metric collection tools mentioned earlier in "Aspects of Capacity Tracking Tools" are frontends to RRDTool.

Ganglia

The charts in this chapter were generated by Ganglia (http://ganglia.info). I had several reasons for choosing this frontend to present examples and illustrate useful monitoring practices. First, Ganglia is the tool we currently use for this type of monitoring at Flickr. We chose it based partly on some general reasons that might make it a good choice for you as well: it's powerful (offering good support for the criteria I listed at the beginning of the chapter) and popular. But in addition, Ganglia was developed originally as a grid management and measurement mechanism aimed at high performance computing (HPC) clusters. Ganglia works well for Flickr's infrastructure because our architecture is similar to HPC environments, in that our backend is segmented into different clusters of machines that each play a different role.
The principles in this chapter, however, are valuable regardless of which monitoring tool you use. Fundamentally, Ganglia works similarly to most metric collection and storage tools. Its metric collection agent is called *gmond* and the aggregation server piece is called *gmetad*. The metrics are displayed using a PHP-based web interface.

SNMP

The Simple Network Management Protocol (SNMP) is a common mechanism for gathering metrics for most networking and server equipment. Think of SNMP as a standardized monitoring and metric collection protocol. Most routers, switches, and servers support it.

SNMP collects and sends more types of metrics than most administrators choose to measure. Because most networking equipment and embedded devices are closed systems, you can't run user-installed applications, such as a metric collection agent like *gmond*. However, as SNMP has long been a standard for networking devices, it provides an easy way to extract metrics from those devices without depending on an agent.

Treating Logs As Past Metrics

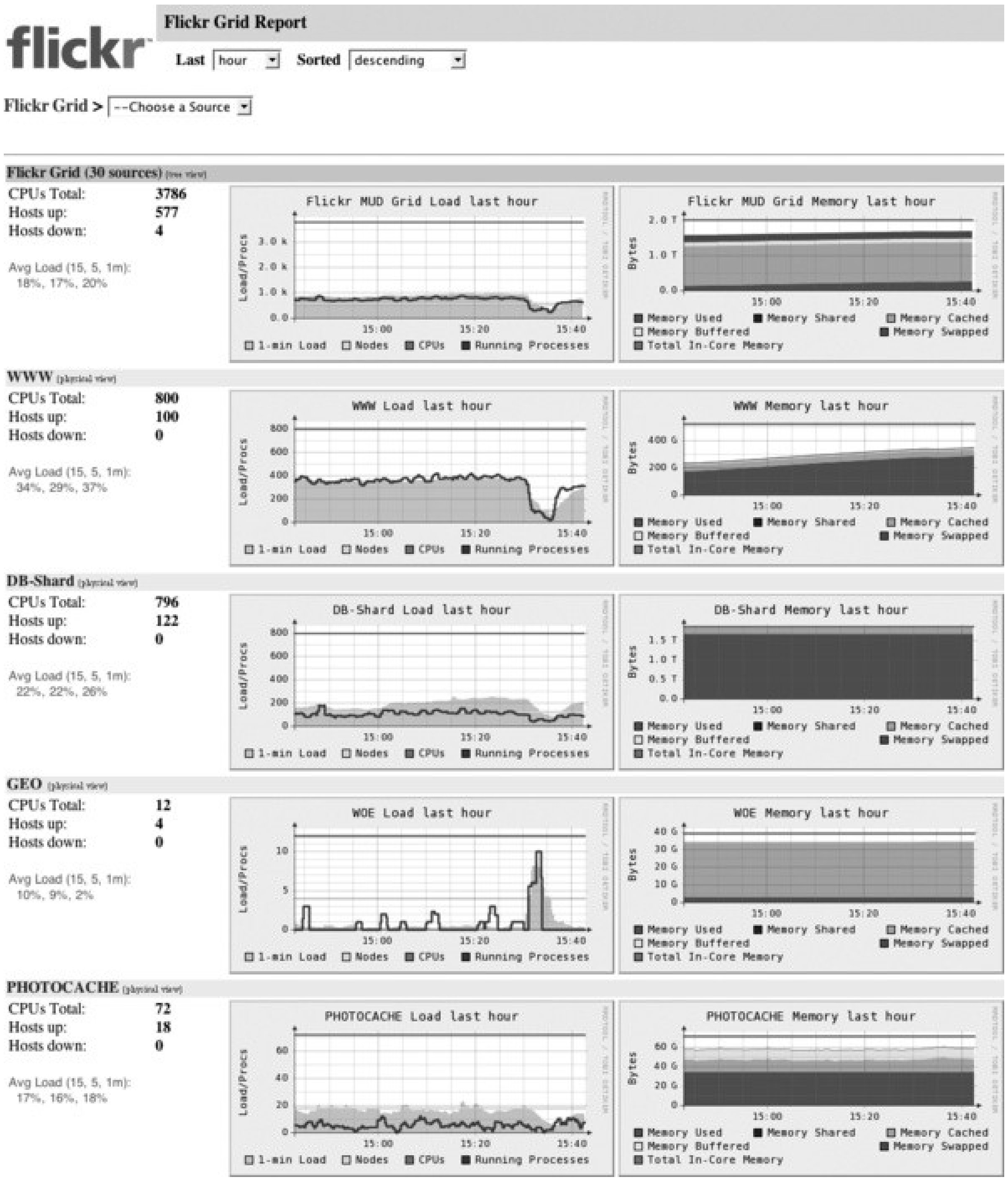
Logs are a great way to inject metrics into your measurement systems, and it underscores one of our criteria for being able to create custom metrics within your monitoring system.

Web servers can log a wealth of information. When you see a spike in resources on a graph, you can often drill down to the access and error logs to find the exact moment those resources jumped. Thus, logs make problem identification easier. Most databases have options to log queries that exceed a certain amount of time, allowing you to identify and fix those slow-running queries. Almost everything you use—mail servers, load balancers, firewalls–has the ability to create logs, either directly or via a Unix-style syslog facility. As an example, at Flickr we count the number of web server error and access log lines per minute and include those metrics into Ganglia's graphs.

Monitoring As a Tool for Urgent Problem Identification

As will be mentioned in the upcoming section, "Applications of Monitoring," problem notification is a separate area of expertise from capacity planning, and generally uses different tools. But some emerging problems are too subtle to trigger health checks from tools such as Nagios. However, the tools we cover in this chapter can be pressed into service to warn you of impending trouble. The techniques in this section can also quickly show you the effects of an optimization.

Figure 3-2 shows some anomalous behavior we once discovered on Flickr through Ganglia. It represents several high-level views of some of Flickr's clusters.



Flickr Grid Report

Last hour Sorted descending

Flickr Grid > --Choose a Source

Flickr Grid (30 sources) (tree view)

CPUs Total:	3786
Hosts up:	577
Hosts down:	4

Avg Load (15, 5, 1m):
18%, 17%, 20%

WWW (physical view)

CPUs Total:	800
Hosts up:	100
Hosts down:	0

Avg Load (15, 5, 1m):
34%, 29%, 37%

DB-Shard (physical view)

CPUs Total:	796
Hosts up:	122
Hosts down:	0

Avg Load (15, 5, 1m):
22%, 22%, 26%

GEO (physical view)

CPUs Total:	12
Hosts up:	4
Hosts down:	0

Avg Load (15, 5, 1m):
10%, 9%, 2%

PHOTOCACHE (physical view)

CPUs Total:	72
Hosts up:	18
Hosts down:	0

Avg Load (15, 5, 1m):
17%, 16%, 18%

Figure 3-2. Using metric collection to identify problems

Without even looking into the details, you can see from the graphs on the left that something unusual has just happened. These graphs cover the load and running processes on the cluster, whereas the groups on the right display combined reports on the memory usage for those clusters. The X axes for all of the graphs correspond to the same time period, so it’s quite easy to see the number of running processes (notably in the WWW cluster) dip in conjunction with the spike in the GEO cluster.

The WWW cluster obviously contains Apache frontend machines serving flickr.com, and our GEO cluster is a collection of servers that perform geographic lookups for features such as photo geotagging. By looking at this one web page, I can ascertain where the problem originated (GEO) and where its effects were felt (all other clusters). As it turns out, this particular event occurred when one of our GEO servers stalled on some of its requests. The connections from our web servers accumulated as a result. When we restarted the GEO server, the web servers gradually recovered.
When faults occur with your website, there is tremendous value in being able to quickly gather status information. You want to be able to get fast answers to the following questions:

- What was the fault?
- When did the fault occur?
- What caused the fault?

In this example, Figure 3-2 helped us pinpoint the source of the trouble because we could correlate the event's effects (via the timeline) on each of our clusters.

Network Measurement and Planning

Capacity planning goes beyond servers and storage to include the network to which they're all connected. The implementation details of routing protocols and switching architectures are not within the scope of this book, but your network is just like any of your other resource: finite in capacity, and well worth measuring.

Networks are commonly viewed as plumbing for servers, and the analogy is apt. When your network is operating well, data simply flows. When it doesn't, everything comes to a grinding halt. This isn't to say that subtle and challenging problems don't crop up with networking: far from it. But for the most part, network devices are designed to do one task well, and their limits should be clear.

Network capacity in hosted environments is often a metered and strictly controlled resource; getting data about your usage can be difficult, depending on the contract you have with your network provider. As a sanity check on your inbound and outbound network usage, aggregate your outward-facing server network metrics and compare them with the bill you receive from your hosting provider.

When you own your own racks and switches, you can make educated decisions about how to divide the hosts across them according to the network capacity they'll need. For example, at Flickr, our photo cache servers demand quite a bit from their switches, because all they do is handle requests for downloads of photos. We're careful not to put too many of them on one switch so the servers have enough bandwidth.

Routers and switches are like servers in that they have various metrics that can be extracted (usually with the SNMP protocol) and recorded. While their main metrics are the bytes *in* and *out* per second (or packets in and out if the payloads are small), they often expose other metrics as well, such as CPU usage and current network sessions.
All of these metrics should be measured on a periodic basis with a network graphing tool, such as MRTG, or some other utility that can store the history for each metric. Unlike Ganglia and other metric collection tools, MRTG is built with SNMP in mind. Simply because your switch and router are well below your limits of network capacity doesn’t mean you’re not nearing CPU usage ceilings on those devices—all of those metrics should be monitored with alerting thresholds as well.

Load Balancing

Load balancers have been a source of much joy and pain in the field of web operations. Their main purpose is to distribute load among pools, or clusters of machines, and they can range from the simplest to the most complex beasts in your data center. Load balancing is usually implemented on the frontend of the architecture, playing traffic cop to web servers that respond to data requests from user’s browsers. But load balancers have also been used to spread load across databases, middle-layer application servers, geographically dispersed data centers, and mail servers; the list continues on.

Load balancers establish load distribution based on a relatively short list of algorithms, and enable you specify the protocols to balance across the available servers serving the traffic. *Scalable Internet Architectures* by Theo Schlossnagle (Pearson) contains some excellent insights into load balancers and their role in web architectures.
For our purposes, load balancers provide a great framework for capacity management, because they allow the easy expansion and removal of capacity in a production environment. They also offer us a place to experiment safely with various amounts of live web traffic so we can track the real effect it has on our server’s resources. You’ll see later why this is useful in helping to find your server’s ceilings. This can be the joy found with load balancing: convenience in deploying and researching capacity.

But there is also pain. Because load balancers are such an integral part of the architecture, failures can be spectacular and dramatic. Not all situations call for load balancing. Even when load balancing is needed, not all balancing algorithms are appropriate.

Jeremy Zawodny recounted a story in the first edition of *High Performance MySQL* (O’Reilly) in which databases at Yahoo! were being load balanced with a “least connections” scheme. This scheme works quite well when balancing web servers: it ensures the server with the smallest number of requests has more traffic directed to it. The reason it works with web servers is web requests are almost always short-lived and on average don’t vary to a great extent in size or latency. The paradigm falls apart, however, with databases because not all queries are the same in terms of size and time to process, and the results of those queries can be quite large. The lesson Zawodny leaves us is just because a database has relatively few current connections does not mean it can tolerate more load.

A second concern with load balancing databases is how to check the health of specific servers within the pool to determine if they all remain capable of receiving traffic. As mentioned earlier, databases are application-specific beasts, so what will work for my application might not work for yours. For me, replication slave lag may be the determining factor for health, whereas for you, it could be the current rate of `SELECT` statements.

Further complications in load balancing include uncommon protocols, complicated balancing algorithms, and the tuning needed to ensure load balancing is working optimally for your application.

Applications of Monitoring

The remainder of this chapter uses examples to demonstrate some of the important monitoring techniques you need to know and perform.

Application-Level Measurement

As mentioned earlier, server statistics paint only a part of the capacity picture. You should also measure and record higher-level metrics specific to your application—not specific to one server, but to the whole system. CPU and server disk usage on a web server doesn’t tell the whole tale of what’s happening to each web request, and a stream of web requests can involve multiple pieces of hardware.

At Flickr, we have a dashboard that collects these application-level metrics. They are collected on both a daily and cumulative basis. Some of the metrics can be drawn from a database, such as the number of photos uploaded. Others can come from aggregating some of the server statistics, such as total disk space consumed across disparate machines. Data collection techniques can be as simple as running a script from a cron job and putting results into its own database for future mining.

Some of the metrics currently tracked at Flickr are:

- Photos uploaded (daily, cumulative)
- Photos uploaded per hour
- Average photo size (daily, cumulative)
- Processing time to segregate photos based on their different sizes (hourly)

- User registrations (daily, cumulative)
- Pro account signups (daily, cumulative)
- Number of photos tagged (daily, cumulative)
- API traffic (API keys in use, requests made per second, per key)
- Number of unique tags (daily, cumulative)
- Number of geotagged photos (daily, cumulative)

We also track certain financial metrics, such as payments received (which lie outside the scope of this book). For your particular application, a good exercise would be to spend some time correlating business and financial data to the system and application metrics you're tracking.

For example, a Total Cost of Ownership (TCO) calculation would be incomplete without some indication of how much these system and application metrics cost the business. Imagine being able to correlate the real costs to serve a single web page with your application. Having these calculations would not only put the architecture into a different context from web operations (business metrics instead of availability, or performance metrics), but they can also provide context for the more finance-obsessed, non-technical upper management who might have access to these tools.

I can't overemphasize the value inherent to identifying and tracking application metrics. Your efforts will be rewarded by imbuing your system statistics with context beyond server health, and will help guide your forecasts. During the procurement process, TCO calculations will prove to be invaluable, as we'll see later.

Now that we've covered the basics of capacity measurement, let's take a look at which measurements you—the manager of a potentially fast-growing website—will likely want to pay special attention. I'll discuss the common elements of web infrastructure and list considerations for measuring their capacity and establishing their upper limits. I'll also provide some examples taken from Flickr's own capacity planning to add greater relevance. The examples are designed to illustrate useful metrics you may wish to track as well. They are not intended to suggest Flickr's architecture or implementation will fit every application's environment.

Storage Capacity

The topic of data storage is vast. For our purposes, I'm going to focus only on the segments of storage that directly influence capacity planning for a high data volume website.

One of the most effective storage analogies is that of a glass of water. The analogy combines a finite limit (the size of the glass) with a variable (the amount of water that can be put into and taken out of the glass at any given time). This helps you to visualize the two major factors to consider when choosing where and how to store your data:

- The maximum capacity of the storage media
- The rate at which the data can be accessed

Traditionally, most web operations have been concerned with the first consideration—the size of the glass. However, most commercial storage vendors have aligned their product lines with both considerations in mind. In most cases, there are two options: large, slow, inexpensive disks (usually using ATA/SATA), and smaller, fast, expensive disks (SCSI and SAS technologies).

Even though the field of data storage has matured, there are still many emerging—and possibly disruptive—technologies of which you should be aware. The popularity of solid-state drives and the hierarchical storage schemes that incorporate them may soon become the norm, as the costs of storage continue to drop and the raw I/O speed of storage has remained flat in recent years.

Consumption rates

When planning the storage needs for your application, the first and foremost consideration should be the *consumption rate*. This is the growth in your data volume measured against a specific length of time. For sites that consume, process, and store rich media files, such as images, video, and audio, keeping an eye on storage consumption rates can be critical to the business. But consumption is important to watch even if your storage doesn't grow much at all.
Disk space is about the easiest capacity metric to understand. Even the least technically inclined computer user understands what it means to run out of disk space.

For storage consumption, the central question is:

When will I run out of disk space?
A real-world example: Tracking storage consumption

At Flickr, we consume a lot of disk space as photos are uploaded and stored. I'll use this simple case as an example of planning for storage consumption.

When photos are uploaded, they are divided into different groups based on size, and sent to a storage appliance. We collect a wide range of metrics related to this process, including:

- How much time it takes to process each image into its various sizes
- How many photos were uploaded
- The average size of the photos
- How much disk space is consumed by those photos

Later, we'll see why we chose to measure these, but for the moment our focus is on the last item: the total disk space consumption over time.

We collect and store this number on a daily basis. The daily time slice has enough detail to show weekly, monthly, seasonal, and holiday trends. Thus, it can be used to predict when we'll need to order more storage hardware. Table 3-1 presents disk space consumption (for photos only) for a two-week period in 2005.

Table 3-1. Sample statistics on daily disk space consumption

Date	Total usage (GB)	Daily usage (GB)
07/26/05	14321.83	138.00
07/27/05	14452.60	130.77
07/28/05	14586.54	133.93
07/29/05	14700.89	114.35
07/30/05	14845.72	144.82
07/31/05	15063.99	218.27
08/01/05	15250.21	186.21
08/02/05	15403.82	153.61
08/03/05	15558.81	154.99
08/04/05	15702.35	143.53

Date	Total usage (GB)	Daily usage (GB)
08/05/05	15835.76	133.41
08/06/05	15986.55	150.79
08/07/05	16189.27	202.72
08/08/05	16367.88	178.60

The data in Table 3-1 is derived from a cron job that runs a script to record the output from the standard Unix `df` command on our storage appliances. The data is then aggregated and included on a metrics dashboard. (We also collect data in much smaller increments [minutes] using Ganglia, but this is not relevant to the current example.)
When we plot the data from Table 3-1, two observations become clear, as shown in Figure 3-3.

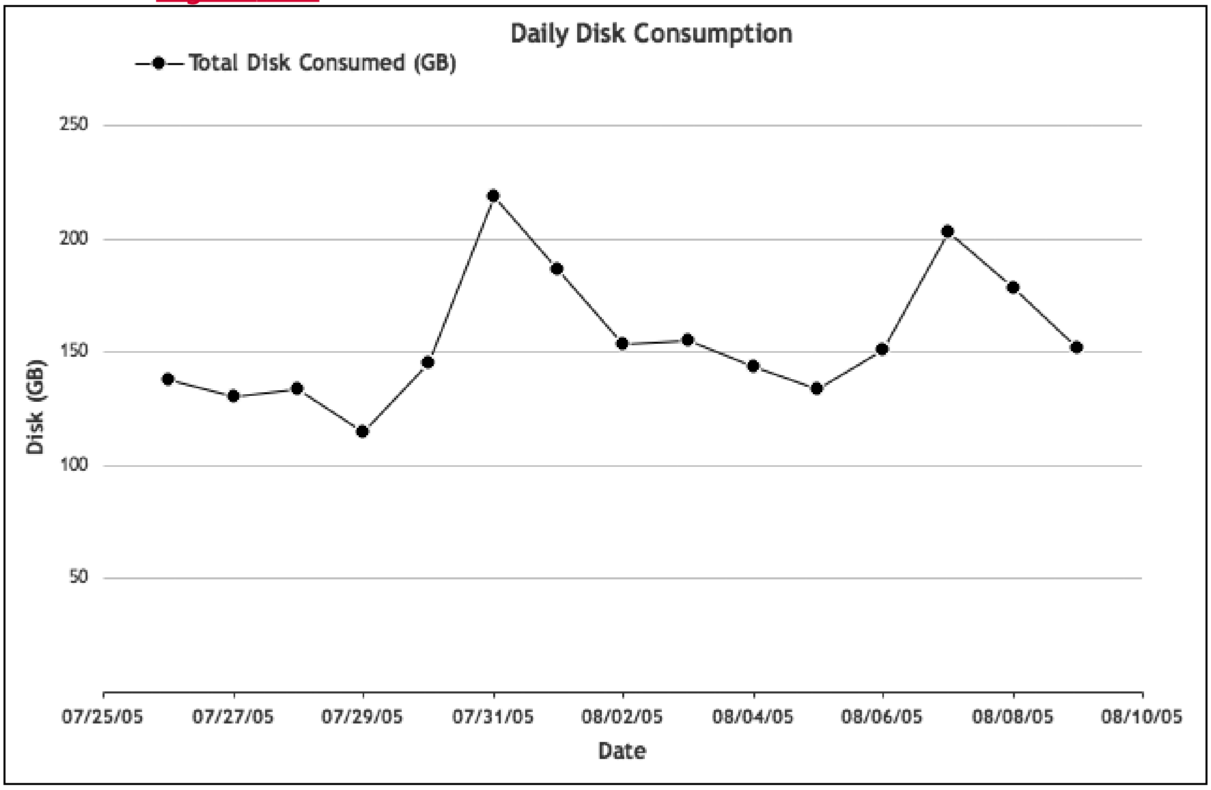


Figure 3-3. Table of daily disk consumption

We can quickly see that the dates 7/31 and 8/07 were high upload periods. In fact, the 31st of July and the 7th of August in 2005 were both

Sundays. Indeed, metrics gathered over a long period reveal Sundays have always been the weekly peak for uploads. Another general trend that can be seen in the chart is Fridays are the lowest upload days of the week. We’ll discuss trends in the next chapter, but for now, it’s enough to know you should be collecting your data with an appropriate resolution to illuminate trends. Some sites show variations on an hourly basis (such as visits to news or weather information); others use monthly slices (retail sites with high-volume periods prior to Christmas).

Storage I/O patterns

How you’re going to access your storage is the next most important consideration. Are you a video site requiring a considerable amount of sequential disk access? Are you using storage for a database that needs to search for fragmented bits of data stored on drives in a random fashion?

Disk utilization metrics can vary, depending on what sort of storage architecture you’re trying to measure, but the basics are:

- How much you’re reading
- How much you’re writing
- How long your CPU is waiting for either reading or writing to finish

Disk drives are the slowest devices in a server. Depending on your server’s load characteristic, these metrics could be what defines your capacity for an entire server platform. Disk utilization and throughput can be measured a number of ways. You’ll find a lot of useful disk measurement tools in Appendix C.

Whether you’re using RAID on a local disk subsystem, a Network-Attached Storage (NAS) appliance, a Storage-Area Network (SAN), or any of the various clustered storage solutions, the metrics you should monitor remain the same: disk consumption and disk I/O consumption. Tracking available disk space and the rate at which you’re able to access that space is irrelevant to which hardware solution you end up choosing; you still need to track them both.

Logs and backup: The metacapacity issue

Backups and logs can consume large amounts of storage, and instituting requirements for them can be a large undertaking. Both backups and logs are part of any sane Business Continuity Plan (BCP) and Disaster Recovery (DR) procedure, so you’ll need to factor in those requirements along with the core business requirements. Everyone needs a backup plan, but for how long do you maintain backup data? A week? A month? Forever? The answers to those questions will differ from site to site, application to application, and business to business.

For example, when archiving financial information, you may be under legal obligation to store data for a specific period of time to comply with federal regulations. On the other hand, some sites—particularly search engines—typically *maintain* their stored data (such as search logs) for shorter durations in an effort to preserve user privacy.
Storage for backups and log archiving entails identifying how much of each you must have readily available (outlined in your rentention/purge policies) versus how much you can archive off to cheaper (usually slower) hardware. There’s nothing particularly special about planning for this type of storage, but it’s quite often overlooked, as sites don’t usually depend on logging and backups for critical guarantees of uptime. We’ll discuss how you go about measuring for growing storage needs later in this chapter. In the next chapter, we’ll explore the process of forecasting those needs.

Measuring loads on web servers

Web server capacity is application-specific. Generally speaking, web servers are considered frontend machines that accept users’ requests, make calls to backend resources (such as databases) then use the results of those calls to generate responses. Some applications make simple and fast database queries; others make fewer, but more complex queries. Some websites serve mostly static pages, whereas others prepare mainly dynamic content. You’ll use both system and application-level metrics to take a long view of the usage metrics, which will serve as the foundation for your capacity plan.

Capacity planning for web servers (static or dynamic) is peak-driven, and therefore elastic, unlike storage consumption. The servers consume a wide range of hardware resources over a daily period, and have a breaking point somewhere near the saturation of those resources. The goal is to discover the periodic peaks and use them to drive your capacity trajectory. As with any peak-driven resource, you’ll want to find out when your peaks are, and then drill down into what’s actually going on during those cycles.

A real-world example: Web server measurement

As an example, let’s take a look at the hourly, daily, and weekly metrics of a single Apache web server. Figure 3-4 presents graphs of all three time frames, from which we’ll try to pick out peak periods.
The hourly graph reveals no particular pattern, while the daily graph shows a smooth decline and rise. Most interesting in terms of capacity planning is the weekly graph, which indicates Mondays undergo the highest web server traffic. As the saying goes: X marks the spot, so let’s start digging.

First, let’s narrow down what hardware resources we’re using. We can pare this list down further by ignoring resources that are operating well within their limits during peak time. Looking at memory, disk I/O, and network resources (not covered in this chapter), we can see none of them come close to approaching their limits during peak times. By eliminating those resources from the list of potential bottlenecks, we already know something significant about our web server capacity. What’s left is CPU time, which we can assume is the critical resource. Figure 3-5 displays two graphs tracking CPU usage.

Figure 3-5 demonstrates that at peak, with user and system CPU usage combined, we’re just a bit above 50 percent of total CPU capacity. Let’s compare the trend with the actual work that is done by the web server so we can see whether the peak CPU usage has any visible effects on the application layer. Figure 3-6 shows the number of busy Apache processes at each unit of time we measure.

The graphs in Figures Figure 3-5 and Figure 3-6 confirm what you might expect: the number of busy Apache processes proportionally follows the CPU usage. Drilling into the most recent RRD values, at 50.7 percent of total CPU usage (45.20 user + 5.50 system), we’re engaging 46 Apache processes. If we assume this relationship between CPU and busy Apache processes stays the same—that is, if it’s linear—until CPU usage reaches some high (and probably unsafe) value we haven’t encountered in our graphs, we can have some reasonable confidence that our CPU capacity is adequate.

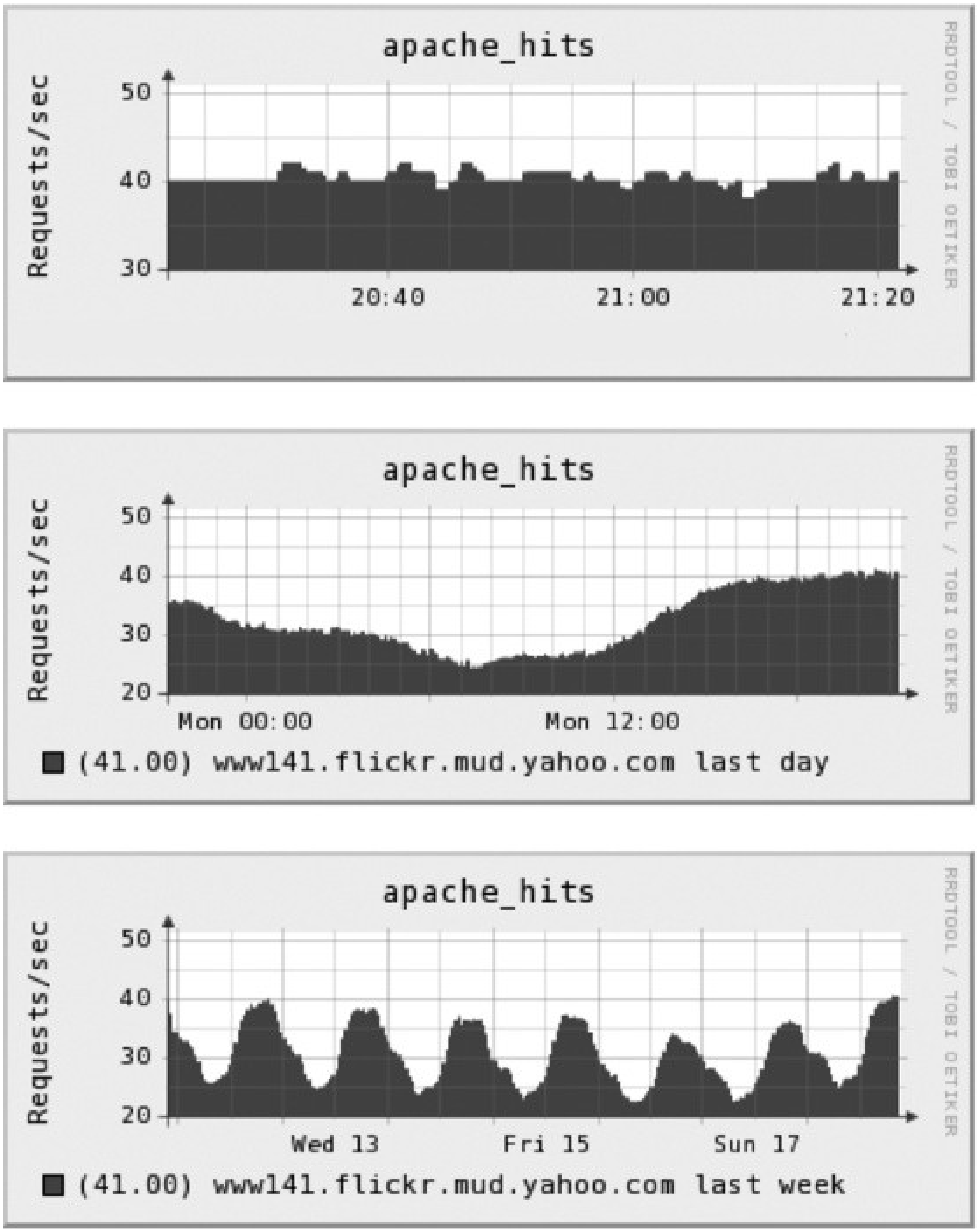
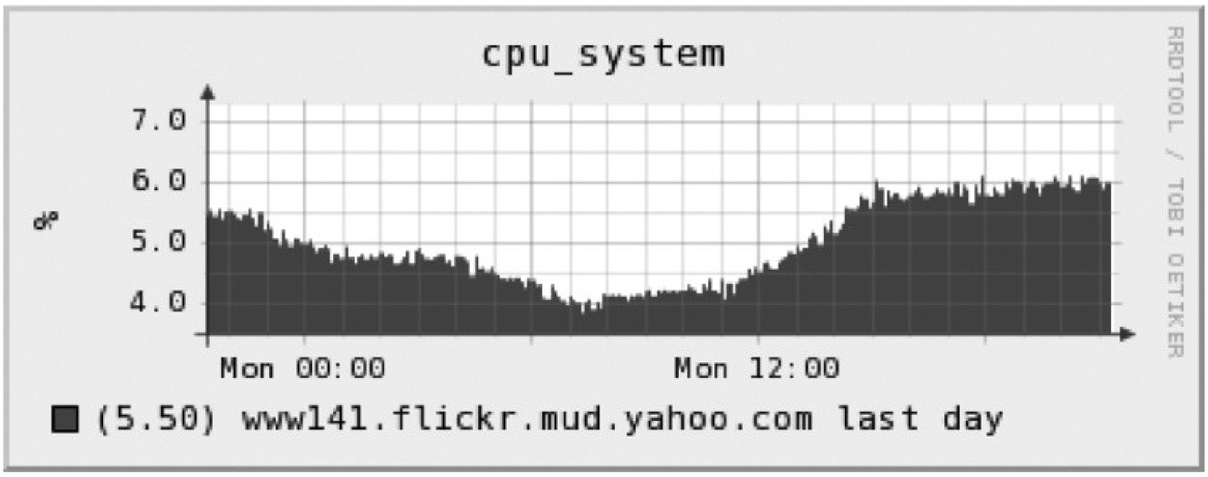


Figure 3-4. Hourly, daily, and weekly view of Apache requests



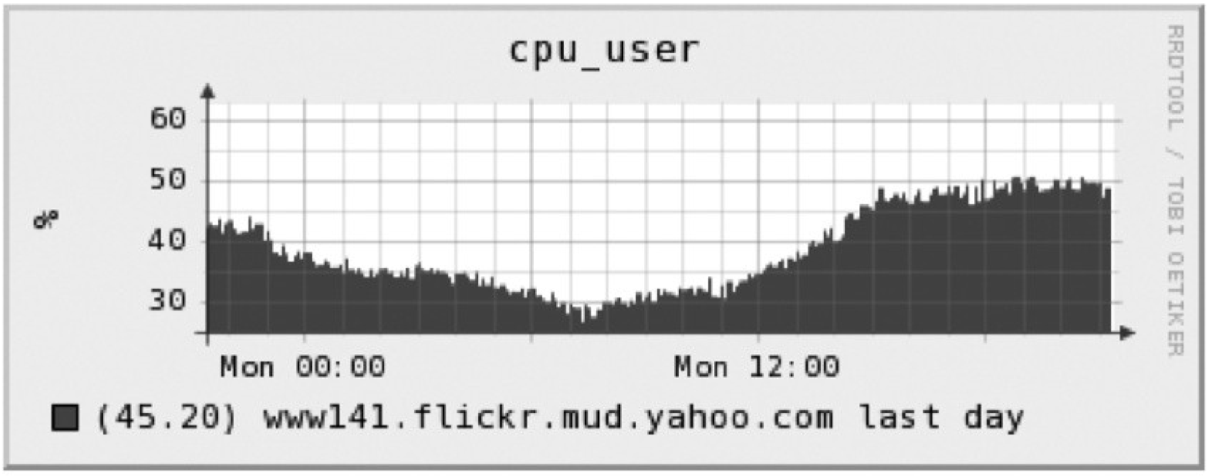


Figure 3-5. User and system CPU on a web server: daily view

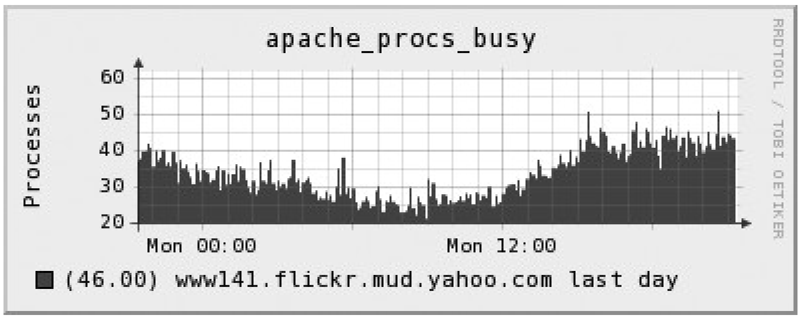


Figure 3-6. Busy Apache processes: daily view

But hey, that's quite an assumption. Let's try to confirm it through another form of capacity planning: *controlled load testing*.

Finding web server ceilings in a load-balancing environment

Capturing the upper limit of your web server's resources can be simplified by at least one design decision that you've probably already made: using a load balancer. In order to confirm your ceiling estimates with live traffic, increase the production load carefully on some web servers and measure the effects it has on resources. Increase the load by pulling machines from the live pool of balanced servers, which increases the load on the remaining servers commensurately. I want to emphasize the importance of using real traffic instead of running a simulation or attempting to model your web server's resources in a benchmark-like setting.

Our artificially load-balanced exercise confirms the assumption we made in the previous section, which is: the relationship between CPU usage and busy Apache processes remain (roughly) constant. Figure 3-7 graphs the increase in active Apache processes and corresponding CPU usage.
This graph was generated from the RRDs produced by one of Flickr's web servers throughout its daily peaks and valleys, sorted by increasing amount of total CPU. It confirms CPU usage does indeed follow the number of busy Apache processes, at least between the values of 40 and 90 percent of CPU.

This suggests at Flickr we are safe using CPU as our single defining metric for capacity on our web servers. In addition, it directly correlates to the work done by the web server—and hopefully correlates further to the website's traffic. Using this information as a basis, we currently set our upper limit on total CPU to be 85 percent, which gives enough headroom to handle occasional spikes while still making efficient use of our servers.

A little more digging shows the ratio of CPU usage to busy Apache processes to be about 1.1, which allows us to compute one from the other. In general, finding a simple relationship between system resource statistics and application-level work will be valuable when you move on to forecasting usage.

ORA: PRODUCTION LOAD TESTING WITH A SINGLE MACHINE

When you have the luxury of using a load balancer to add and remove servers into production, it makes the task of finding capacity ceilings easy. But when you have only a single machine, things become somewhat more difficult. How can you increase load on that one machine?

You can't call the users and ask them to start clicking faster, but you can add load on your server by *replaying* requests you've seen in production. Many tools are available to do this, including two in particular with which I've had good experiences:

- Httperf (http://www.hpl.hp.com/research/linux/httperf/)
- Siege (http://www.joedog.org/JoeDog/Siege)

Both of these tools allow you to take a file filled with HTTP URLs and replay them at varying speeds against your server. This enables you to increase load very slowly and carefully, which is critical in a single-machine environment—you don't want to do any load testing that would kill your only running server. This not-very-safe technique should only be used to stretch the server's abilities a little at a time under safe conditions, such as during a low-traffic period.

Artificial load testing, even by replaying real production logs, brings with it a whole slew of limitations. Even though these tools give fine-grained control over the rate of requests, and you can control what requests are being made, they still don't accurately portray how your server will behave under duress.

For example, depending on how you allow your application to change data, simulating production loads with logs of previous requests can be difficult. You might not want to rewrite data you've already written, so you'll need to do some massaging of the logs you're replaying.

Another limitation is common to all web load-testing scenarios: an accurate representation of the client/server relationship. By using Httperf or Siege, you can simulate multiple clients, but in reality all the requests will be coming from a single client server that's running the script. A workaround for running these tools from a single machine is to run the scripts from multiple machines. A tool called Autobench (http://www.xenoclast.org/autobench/) is a Perl wrapper around Httperf that enables it to be run from multiple hosts in a coordinated fashion. Autobench can also aggregate the results for you.
But even Autobench, produces different results from real-world requests. Clients can be all over the globe, with wildly different network latencies, connection speeds, and many other variables.

Also be careful about cranking up the request rate from a single client machine. You might run out of client resources before you run out of server resources, which will further taint your results. Be sure to stay below the client's file descriptor maximums, network limits, and CPU limits during these artificial tests.

Instead of spending time working around the limitations of artificial testing, you should consider upgrading your architecture to make it easier to find your server's upper limits. A load balancer—even with just one extra server—not only helps you find limits but provides a more robust architecture overall.

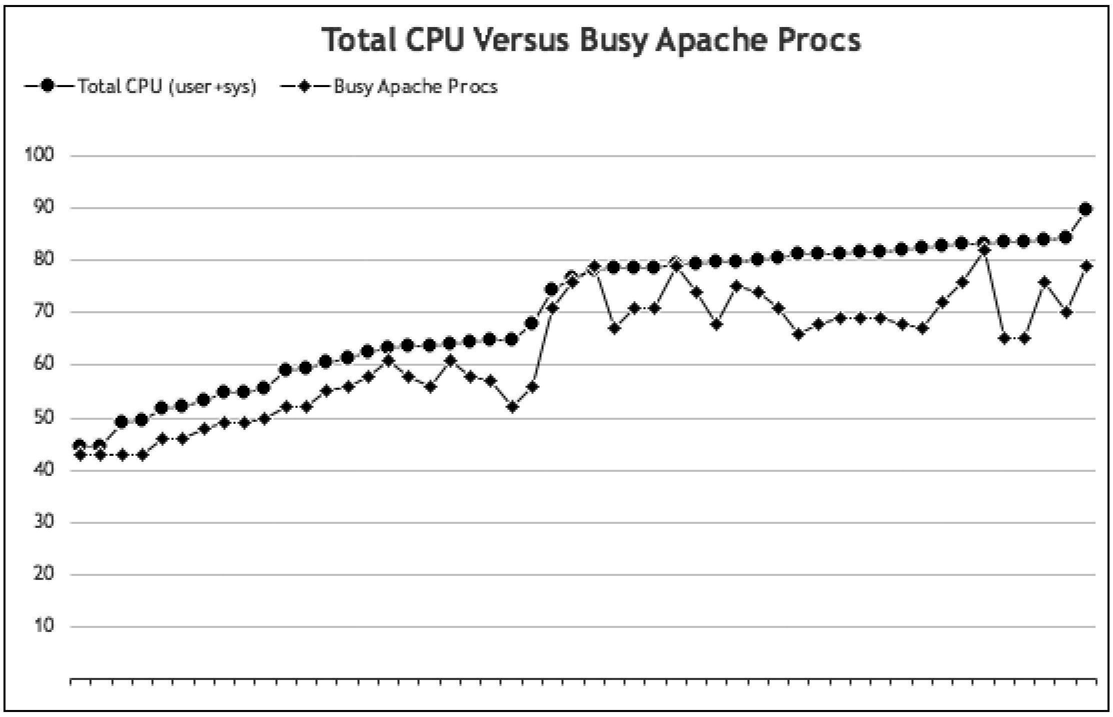


Figure 3-7. Total CPU versus busy Apache processes

Database Capacity

Nearly every dynamic website uses some form of a database to keep track of its data. This means you need to provide the capacity for it. In the LAMP world, MySQL and Postgres are favorite databases, while Oracle, Microsoft SQL server, and a myriad of others also serve as the backend data store for many successful sites.

Outside of the basic server statistics, there are a number of database-specific metrics you'll want to track:

- Queries-per-second (SELECTs, INSERTs, UPDATEs, and DELETEs)
- Connections currently open
- Lag time between master and slave during replication
- Cache hit rates

Planning the capacity needs of databases—particularly clusters of them—can be tricky business. Establishing the performance ceilings of your databases can be difficult because there might be hidden cliffs that only reveal themselves in certain edge cases.

For example, in the past we've made assumptions at Flickr that our databases running on a given hardware platform had a ceiling of *X* queries-per-second before performance began to degrade unacceptably. But we were surprised to learn that some queries perform fine on a user with fewer than 10,000 photos, but slow down alarmingly on a user who has more than 100,000 photos. (Yes, some Flickr users do upload hundreds of thousands of photos there!) So, we redefined our ceilings for the database server that handles users with large numbers of photos. This type of creative sleuthing of capacity and performance is mandatory with databases, and underscores the importance of understanding how the databases are actually being used outside the perspective of system statistics.

At this stage I'll reiterate my point about performance tuning. As pointed out in Jeremy Zawodny and Derek Balling's book, *High Performance MySQL*, database performance often depends more on your schemas and queries than on the speed of your hardware. Because of this, developers and database administrators focus on optimizing their schemas and queries, knowing that doing so can change the performance of the database quite dramatically. This in turn, affects your database ceilings. One day you think you need 10 database servers to get 20,000 queries-per-second; the next day you find you'll only need five, because you (or your programmers) were able to optimize some of the more common (or computationally expensive) queries.

A real-world example: Database measurement

Databases are complex beasts, and finding the limits of your database can be time consuming, but well worth the effort. Just as with web servers, database capacity tends to be peak-driven, meaning their limits are usually defined by how they perform during the heaviest periods of end-user activity. As a result, we generally take a close look at the peak traffic times to see what's going on with system resources, and take it from there.

But before we start hunting for the magical "red line" of database consumption, remember, I recommend looking at how your database performs with *real* queries and *real* data.
One of the first things you should determine is when your database is expected to run out of hardware resources, relative to traffic. Depending on the load characteristics, you might be bound by the CPU, the network, or disk I/O.

If you are lucky enough to keep your most-requested data in memory, you might find yourself being constrained by CPU or network resources. This situation makes your hunt for a performance ceiling a bit easier as you need only track a single number, as we discovered when monitoring Apache performance.

If your data is substantially larger than what you can fit into physical memory, your database's performance will be limited by the slowest piece of equipment: your physical disk. Because of the random nature of database-driven websites, queries for data on databases tend to be yet even more random, and the resulting disk I/O is correspondingly random. Random disk I/O tends to be slow, because the data being requested is bound by the disk's ability to seek back and forth to random points on the platter. Therefore, many growing websites eventually have disk I/O as their defining metric for database capacity.

As it turns out, our databases at Flickr are currently in that position. We know this by taking even a cursory look at our disk utilization statistics, in conjunction with the fact that the data requested from MySQL is much larger than the amount of physical memory we have. Let's use one of the servers as an example. Figure 3-8 shows the relevant MySQL metrics for a single Flickr user database during a peak hour.
Figure 3-8 depicts the rate of concurrent MySQL connections along with the rate of INSERTs, UPDATEs, DELETEs, and SELECTs per second for a single hour. There are a few spikes during this time in each of the metrics, but only one stands out as a potential item of interest.

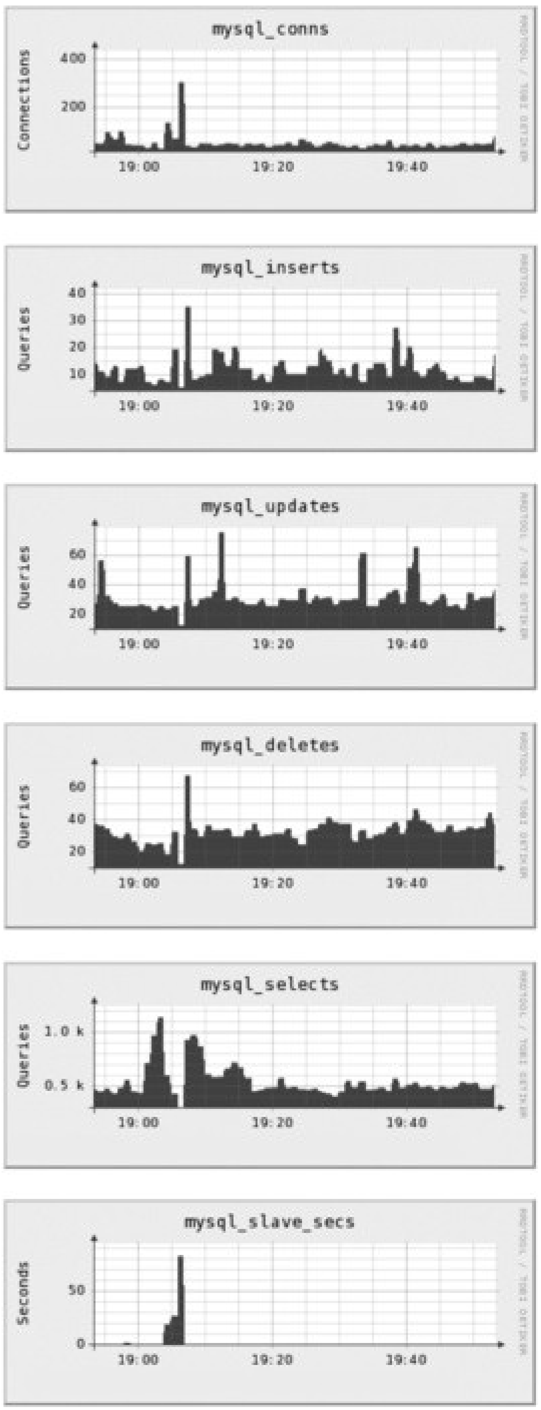
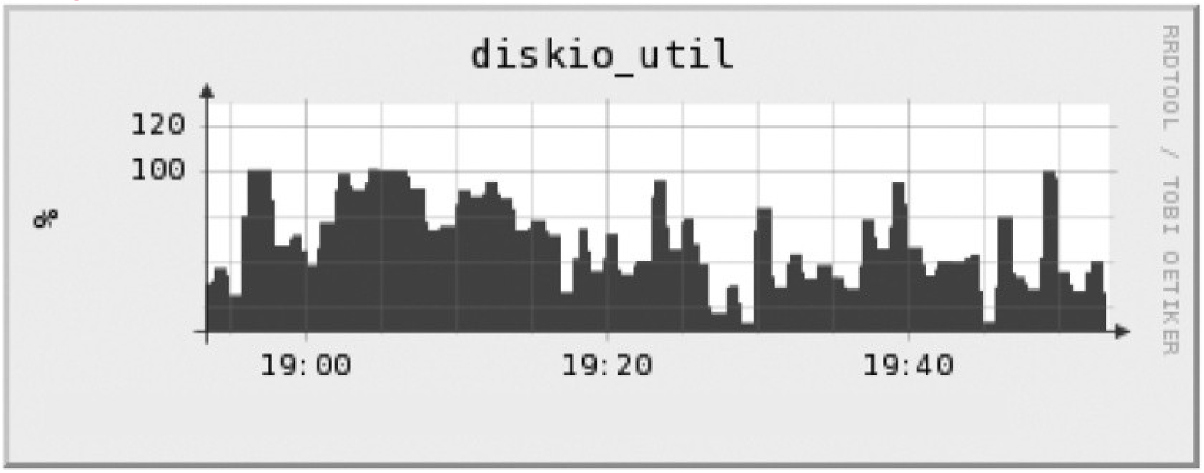


Figure 3-8. Production database MySQL metrics

The bottom graph shows the amount of database replication lag experienced during the last hour; it peaks at over 80 seconds. We don't like to see that degree of lag in database replication because it generally means the slaves temporarily lack a lot of the recent data loaded onto the master. Flickr directs all user queries to the slaves, which means until the slaves catch up to the master, users won't see the most up-to-date data. This can cause various unwelcome effects, such as a user annotating a photo, clicking the Submit button, but not seeing that comment right away. This is confusing for the user and can result in all sorts of out-of-sync weirdness. It's not ideal, to say the least.

From past experience, I’m aware our databases are disk I/O bound, but let’s confirm that by taking a look at disk utilization and I/O wait in Figure 3-9.



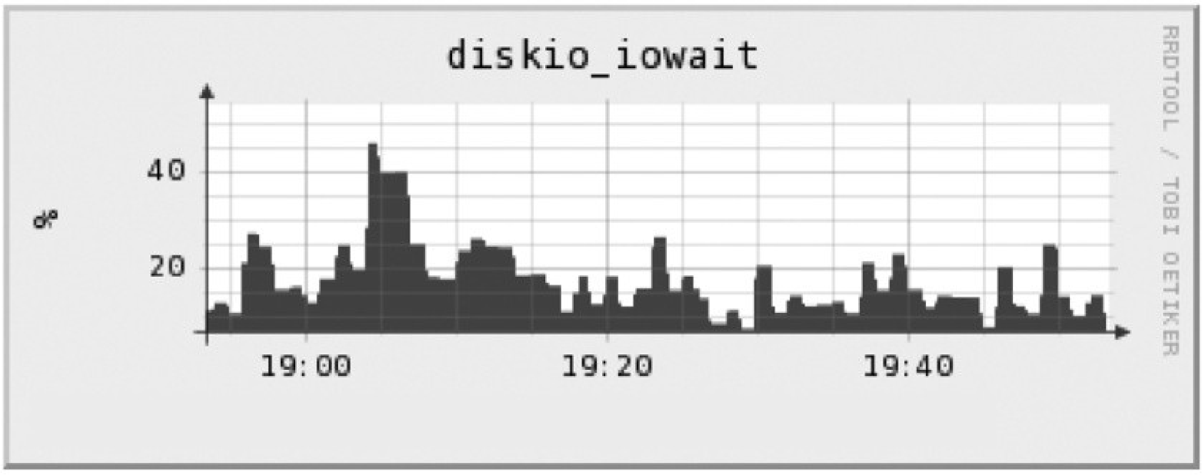


Figure 3-9. Production database disk utilization and I/O wait

In this example, we have Ganglia collecting and graphing disk utilization statistics for our database. These are reported every 60 seconds by some of the fields returned by the Linux *iostat* command, `%iowait` and `%ioutil`. Note on the graph that while disk utilization jumped up to 100 percent more than once, it was only during the period where I/O wait bumped over the 40 percent mark that the MySQL replication lag jumped.

What does this mean? With nothing more than a cursory look at our metrics, we can deduce that replication slave lag is caused by disk I/O wait rather than disk utilization. We can further deduce that the replication lag becomes a problem only at a disk I/O wait of 40 percent or higher. Bear in mind these results apply only to a particular configuration at Flickr; this is an example rather than a general rule. The results make me wonder: could they indicate something defective with this particular server? Possibly, and the hypothesis should be easy enough to verify by provoking the behavior on similar production hardware. In this case, my examination of graphs from other servers demonstrates the relationship to be a general one that applies to Flickr activity at the time: other databases of identical hardware in production experience replication lag starting in, or near 40 percent disk I/O wait.

ORA: CURIOSITY KILLED THE CAPACITY PLAN

Let’s take a moment to address a question that may have arisen in your mind: if the spike is not specific to a hardware defect, and is indeed due to a legitimate database event, what is triggering it? This is a pertinent question, but its answer won’t get you any further in assessing how many database servers you’ll need to handle your traffic. There will always be spikes in resource usage, bugs, bad queries, and other unforeseen hiccups. Your job as a capacity planner is to take the bad with the good, and assume the bad won’t go away. Of course, after you’ve submitted your hardware purchase justifications, by all means, don your performance-tuning hat and go after the cause of that spike with a vengeance. Indeed, finding the cause should be a mandatory next step, just don’t let the investigation of one anomaly get in the way of forecasting capacity needs.

Armed with these metrics, we now we have a degree of confidence the 40 percent disk I/O wait threshold is our ceiling for this database. As it relates to hardware configuration and our database query mix and rate, we should plan on staying below 40 percent disk I/O wait. But what does that mean in terms of actual database work?

Before we dig a bit further into the numbers, let’s apply the same test method as we did with the web servers: increase production load on the database.

Finding database ceilings

A more focused and aggressive approach to finding database ceilings is to slowly (but again, carefully) increase the load on a live production server. If you maintain only a single database, this can be difficult to do safely. With only a single point of failure, testing runs the risk of bringing your site down entirely. This exercise becomes markedly easier if you employ any sort of database load balancing (via hardware appliance or within the application layer). In Figure 3-10, let’s revisit the diagram of a common database architecture, this time with more real-world details added.
In this scenario, all database write operations are directed to the master; read functions are performed by database slaves. The slaves are kept up to date by replication. To pinpoint the ceilings of your database slaves, you want to increase the load on one of them by instructing your application to favor that particular device. If you operate a hardware load balancer for your databases, you may be able to weight one of the servers higher than the others in the balanced pool.

Increasing load to a database in this manner can reveal the effects load has on your resources, and hopefully expose the point at which your load will begin to affect replication lag. In our case, we’d hope to confirm our educated guess of 40 percent disk I/O wait is the upper limit the database can withstand without inducing replication lag.

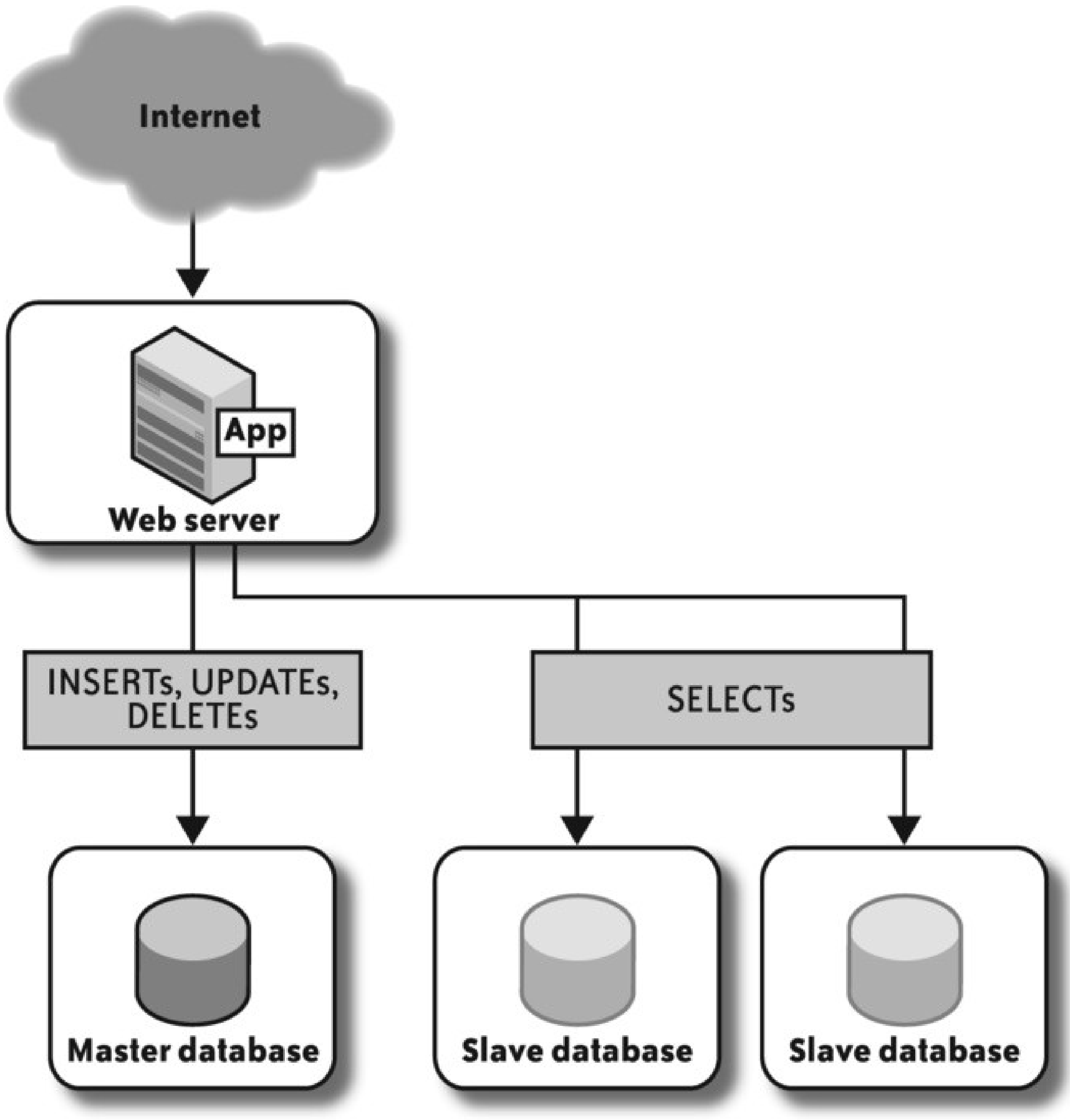


Figure 3-10. A master-slave database architecture

This example reflects a common database capacity issue defined by disk I/O. Your databases might be CPU-, memory-, or network-bound, but the process of finding the ceiling of each server is the same.

Caching Systems

We mentioned earlier that disks are the slowest pieces of your infrastructure, which makes accessing them expensive in terms of time. Most large-scale sites alleviate the need for making these expensive operations by caching data in various locations.

In web architectures, caches are most often used to store database results (as with Memcached) or actual files (as with Squid or Varnish). Both approaches call for the same considerations with respect to capacity planning. They are examples of *reverse proxies*, which are specialized systems that cache data sent from the web server to the client (usually a web browser).

First let’s take a look at the diagram in Figure 3-11 to see how Squid and Varnish caching normally works with servers.

As Figure 3-12 shows, the diagram differs only slightly when illustrating database caching in the style of Memcached.

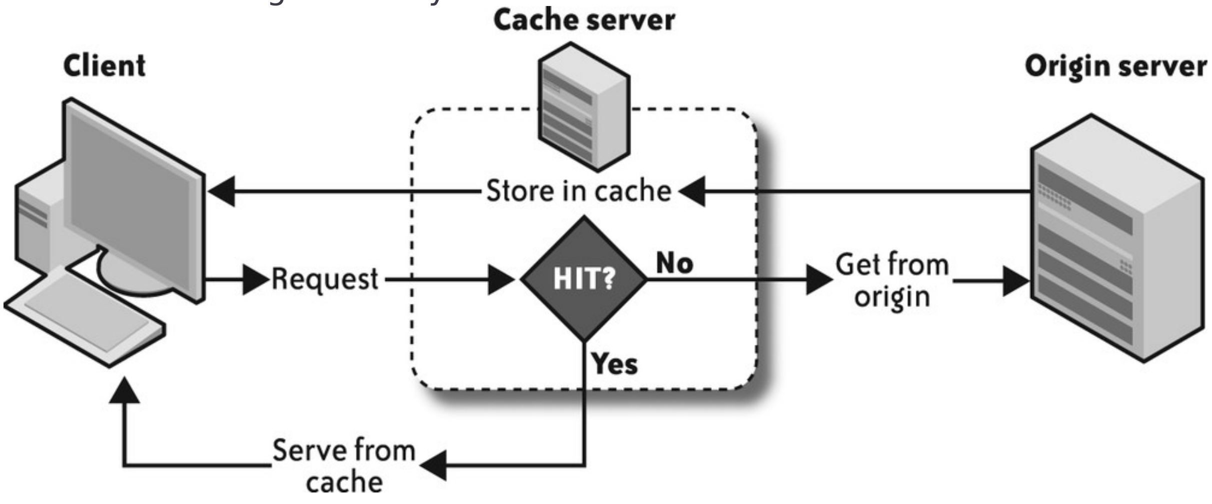


Figure 3-11. Basic content server caching mechanisms (reverse-proxy)

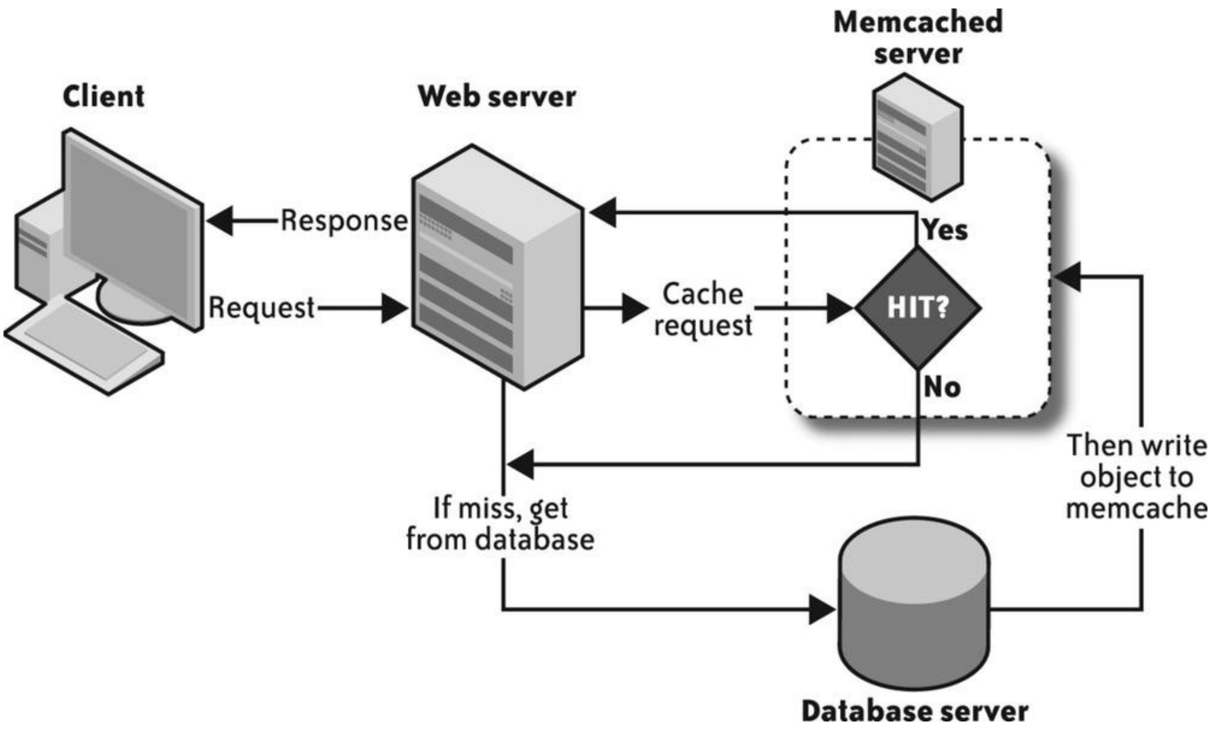


Figure 3-12. Database caching

Cache efficiency: Working sets and dynamic data

The two main factors affecting cache capacity are the size of your *working set* and the extent to which your data is dynamic or changing.
How often your data changes will dictate whether you'll choose to cache that data. On one end of the spectrum is data that almost never changes. Examples of this type of data include user names and account information. On the other end of the spectrum is information that changes frequently, such as the last comment made by a user, or the last photo uploaded. Figure 3-13 illustrates the relationship between caching efficiency and types of data.
It should be obvious there's no benefit in caching data that changes frequently, because the proxy will spend more time invalidating the cache than retrieving data from it. Every application will have its own unique characteristics with respect to caching, so there isn't any rule of thumb to follow. However, measuring and recording your cache's hit ratio is imperative to understanding how efficient it is. This can help guide your capacity plan and hopefully steer how (and when) you'll want to cache objects.

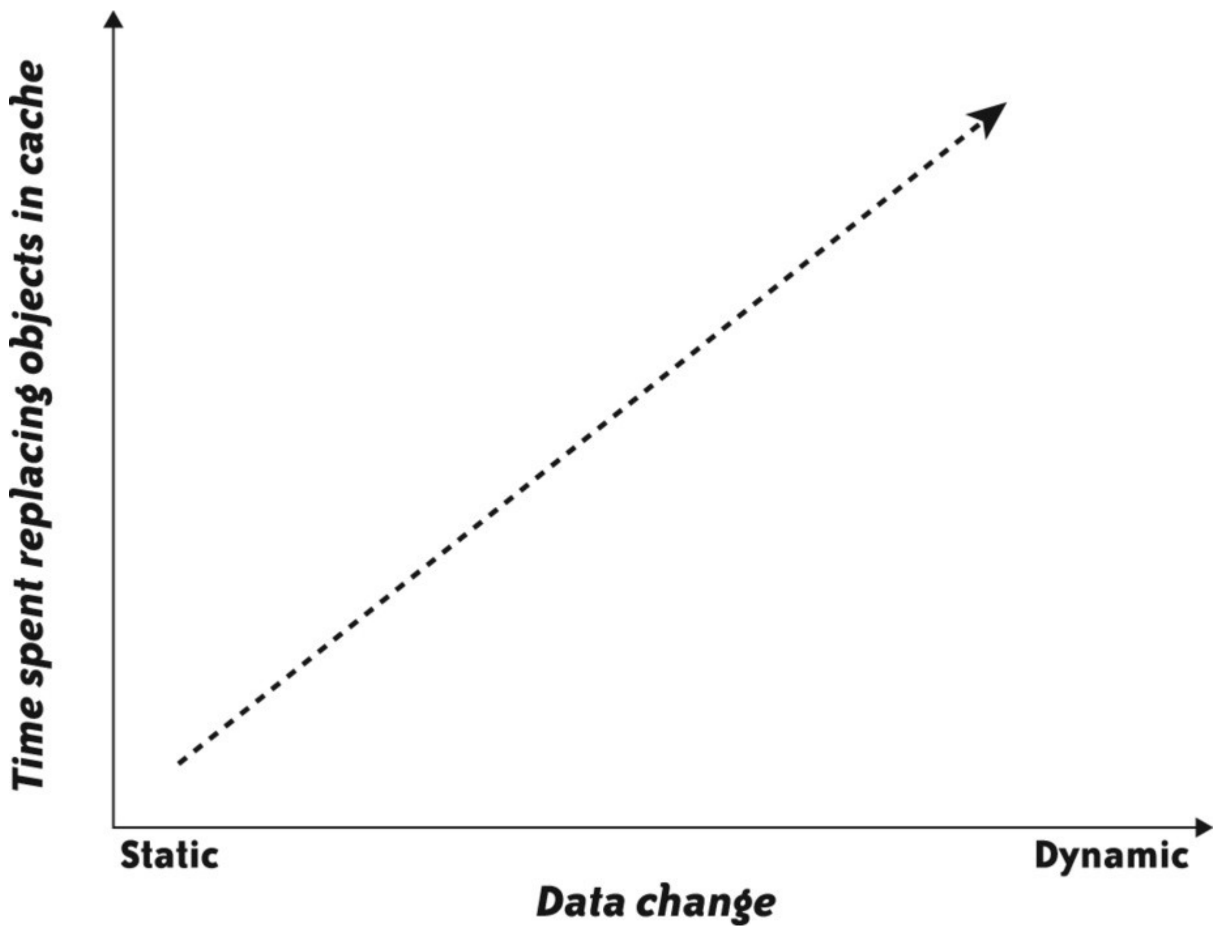


Figure 3-13. Cache efficiency depends on rate of change

The other major consideration is the size of your *working set* of cacheable objects. Caches have a fixed size. Your working set of cacheable objects is

the number of unique objects—whether database results or files—requested over a given time period. Ideally, you'll have enough cache capacity to handle your entire working set. This would mean the vast majority of requests to the cache would result in cache hits. However, in real terms, there could be many reasons why you can't keep all the objects you want in cache. You would then need to depend on something called *cache eviction* to make room for new objects coming in. I'll present more on cache eviction later.
In order to function, caching software needs to keep track of its own metrics internally. Because of this, most proxies expose those metrics, allowing them to be measured and recorded by your monitoring tools.

At Flickr, we use Squid for reverse-proxy caching of photos. We use slower, cheaper, and larger capacity disks to store the photos, but we employ caching systems that use smaller and faster disks to serve those photos. We horizontally scale the number of caching servers as the request rate for photos increases. We also horizontally scale the backend persistent storage as the number of photos grows.

Each caching server has a limited amount of disk and memory available to use as cache. Because our working set of photos is too large to fit into our caches, the caches fill up. A full cache needs to constantly make decisions on which objects to evict to make room for new objects coming in. This process is based on a replacement or "eviction" algorithm. There are many eviction algorithms, but one of the most common is Least Recently Used (LRU), which is demonstrated in Figure 3-14.

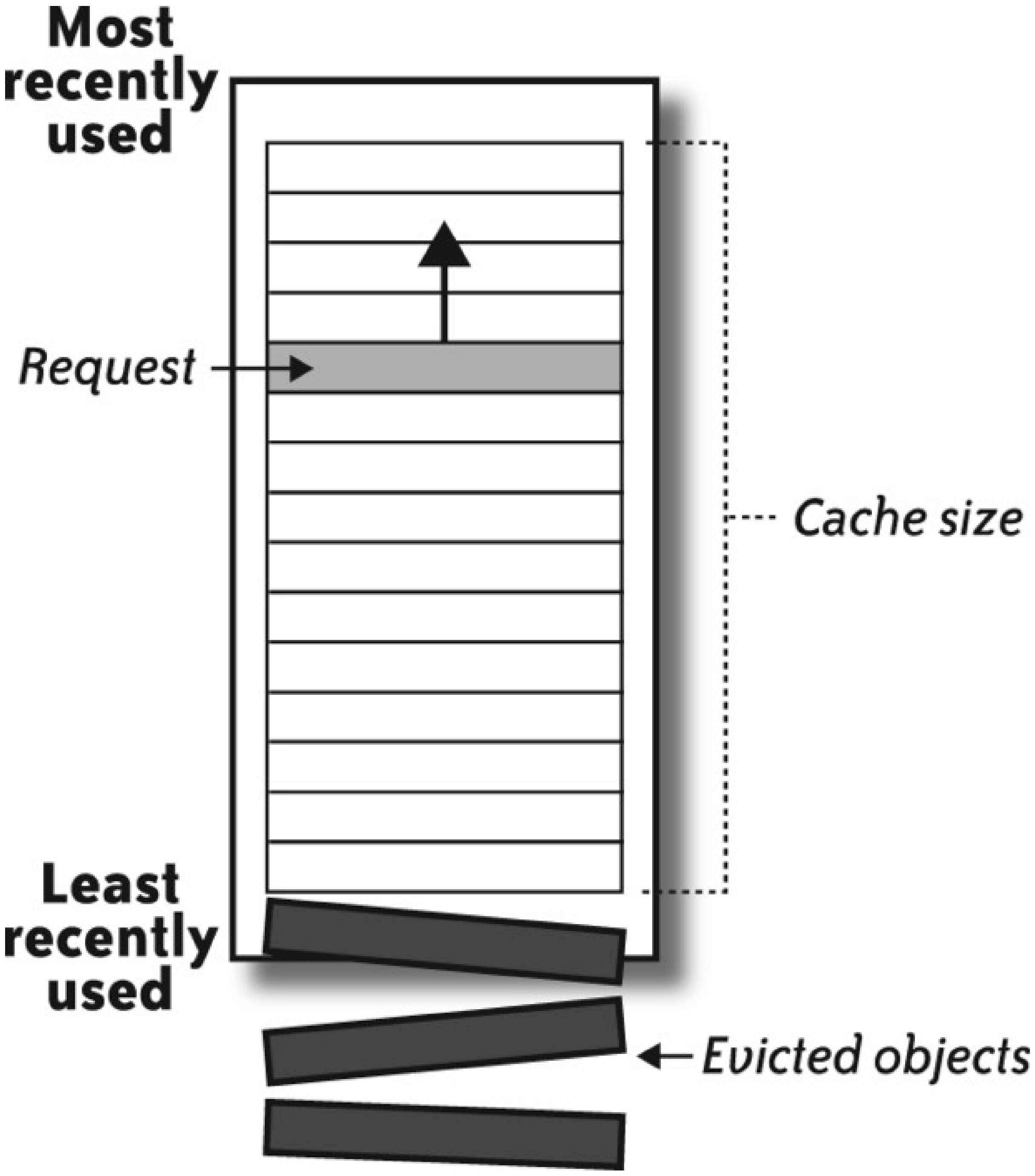


Figure 3-14. The LRU cache eviction algorithm

As requests come into a cache, the objects are organized into a list based on when each was last requested. A cache-missed object, once retrieved from the origin server, will be placed at the top of the list, and a cache hit will also be moved from its current location to the top of the list. This keeps all of the objects in order from most recently used to least recently used. When the cache needs to make room for new objects, it will remove objects from the bottom of the list. The age of the oldest object on the list is known as the *LRU reference age*, and is an indicator of how efficient the cache is, along with the hit ratio.

The LRU algorithm is used in Memcached, Squid, Varnish, and countless other caching applications. Its behavior is well known, and relatively simple to understand. Squid offers a choice of some more complex eviction algorithms, but nearly all of the most popular database caches use the LRU method.

The most important metrics to track with any caching software are:

- Cache hit ratio
- Total request rate
- Average object size
- LRU reference age (when using the LRU method)

Let's take a look at some caching metrics (compiled using Squid) from production.

Establishing Caching System Ceilings

The capacity of caching systems is defined differently depending on their usage. For a cache that can hold its entire working set, the request rate and response time might dictate its upper limits. In this case, you can again make use of the same method we applied to web serving and database disk I/O wait: carefully increase the load on a server in production, gather metric data along the way, and tie system resource metrics (CPU, disk I/O, network, memory usage) to the caching system metrics listed in the previous section.

Determining the ceiling of a cache when it is constantly full and must continually evict objects is a complicated exercise. It may better be defined not by request rate, but by its hit ratio (and indirectly, its reference age).

Table 3-2 summarizes cache planning considerations.

Table 3-2. Cache planning considerations

Type of cache use	Characteristics	Cache ceilings	Resource ceilings
Small, or slowly increasing working set	100% contained in cache	Request rate	Disk I/O utilization and wait, CPU and memory usage
Large, or growing working set	Moving window, constant eviction (churn)	Hit ratio, LRU reference age	Cache size

A real-world example: Cache measurement

As I mentioned earlier, at Flickr we need to take into account all of the metrics mentioned in the previous section. Our caches are constantly full, and cache eviction is a continuous process as users regularly upload new photos. We make use of Squid's memory and disk cache, so both of those resources need to be measured as well.

Let's first take a look at the graphs in Figure 3-15 to see what effect request rates are having on our system resources.
As you can see from the graphs, the Squid request rate increased steadily within the time period shown. The zigzag pattern represents the weekly peak activity periods (Mondays) we discovered earlier. For the same time period, the total CPU usage has increased as well, but we're not in any immediate danger of running out of CPU resources. Because we make extensive use of disk cache on our Squid servers, we'll want to take a look our disk I/O usage as well. See Figure 3-16 for the results.
Figure 3-16 confirms what we suspected: the amount of operations waiting for disk activity follows the rate of requests in near perfect synchronization. Since we know our Squid server uses the disk more than any other resource, such as CPU or memory, this tells us the defining peak resource metric is disk I/O wait—the same as our database ceiling. If we zoom into the data using RRDTool to overlay disk I/O wait and our request rate, we can plot them against each other in Excel, as shown in Figure 3-17.
Now that we have our correlation, let's sort the data in Excel as a function of increasing order and plot it again. As illustrated in Figure 3-18, this permits us to more easily see the trends related to the amount of the data requested at a particular moment.
Now we can clearly see how the two metrics relate to each other as they go up, and how disk I/O wait affects Squid's performance.

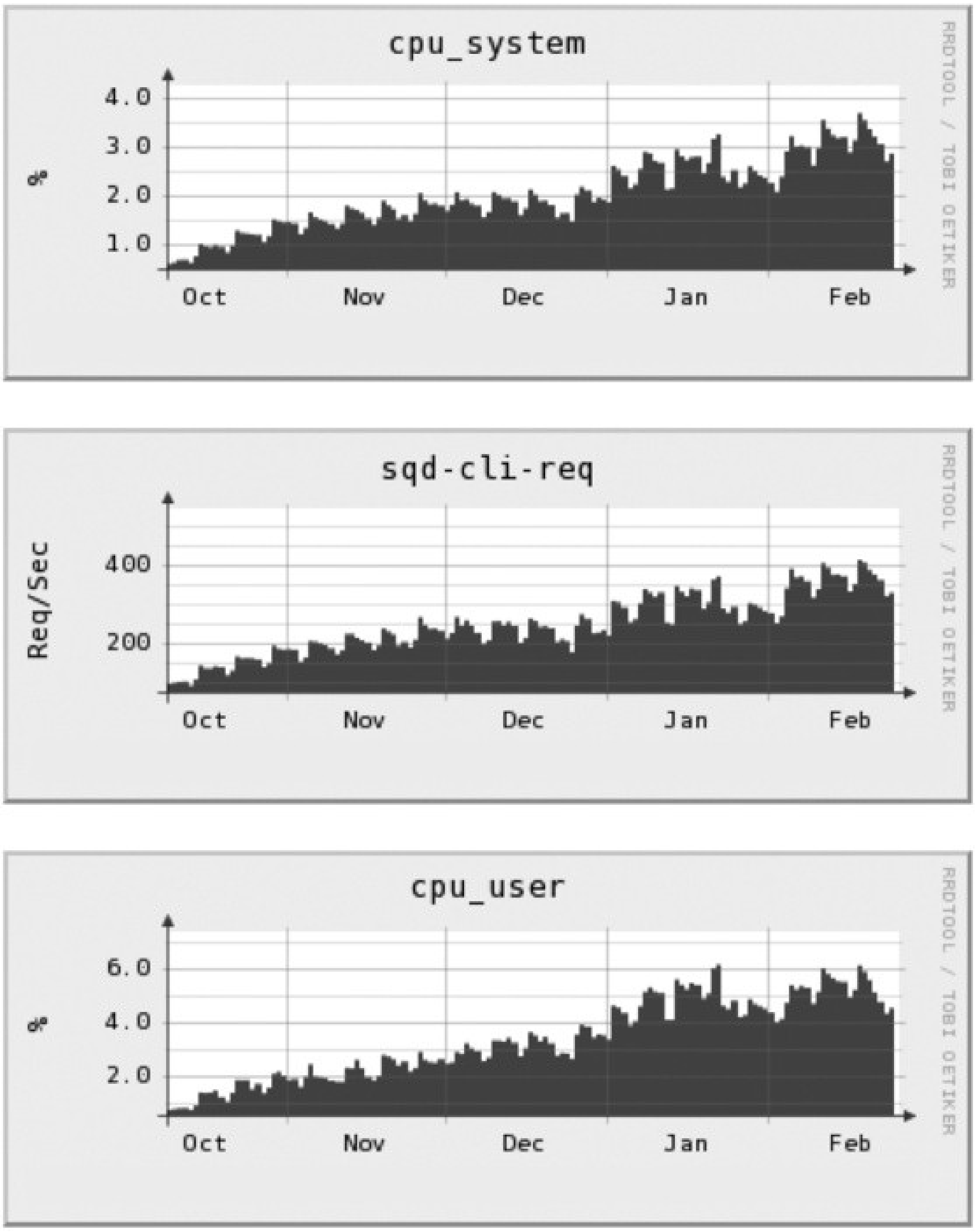
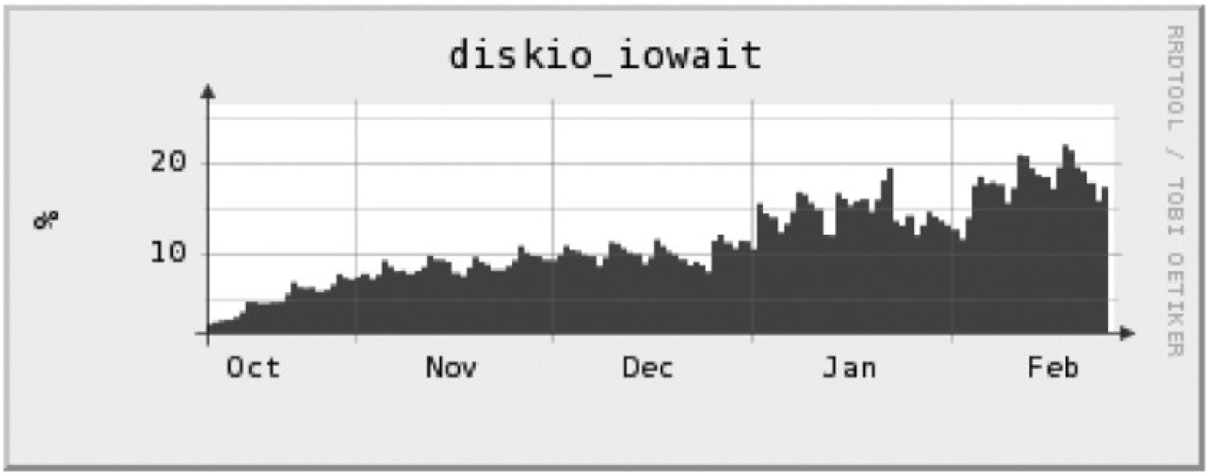


Figure 3-15. Five-month view of Squid request rate and CPU load (user and system)



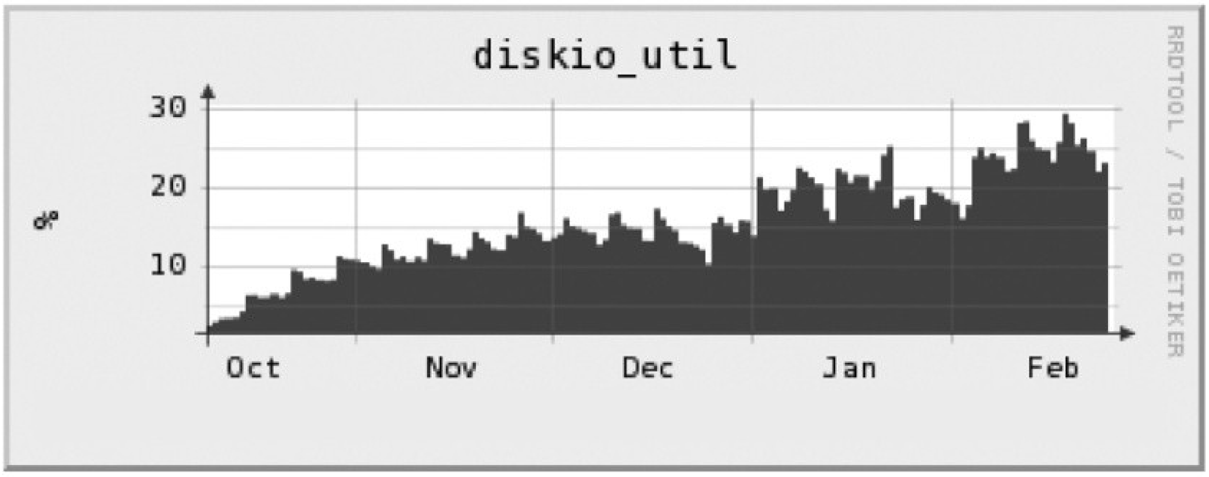


Figure 3-16. Five-month view of Squid server disk I/O wait and utilization

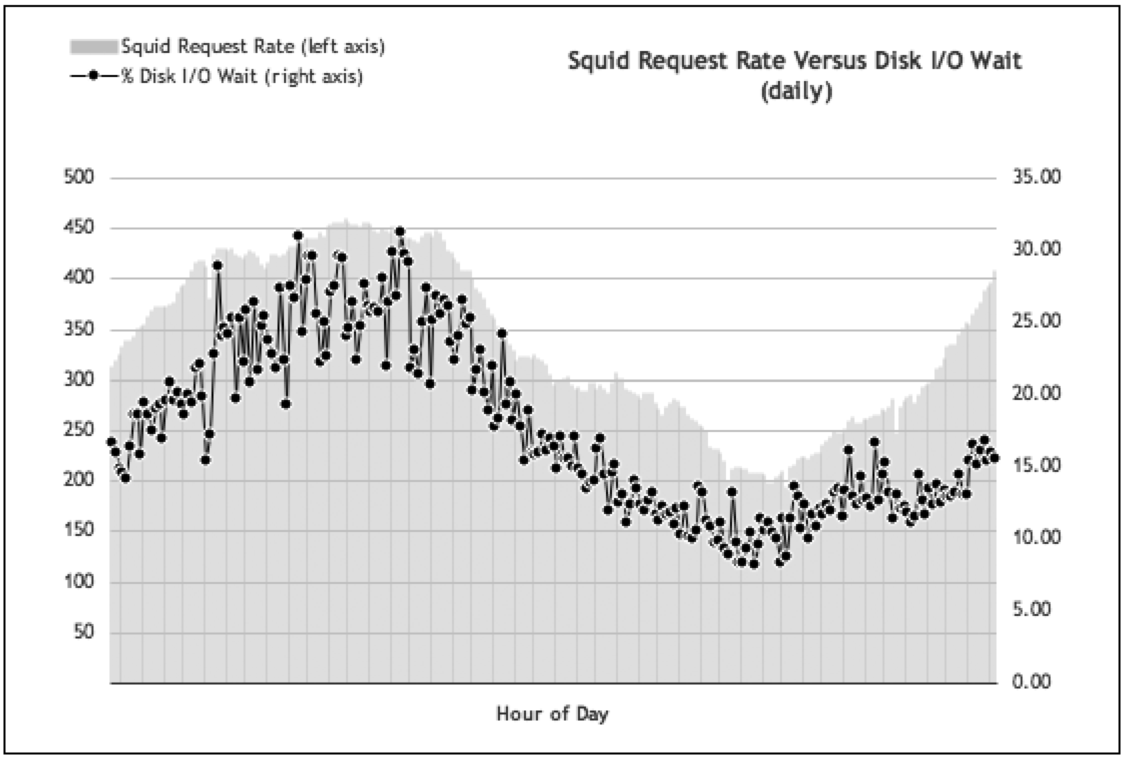


Figure 3-17. Squid request rate versus disk I/O wait: daily view

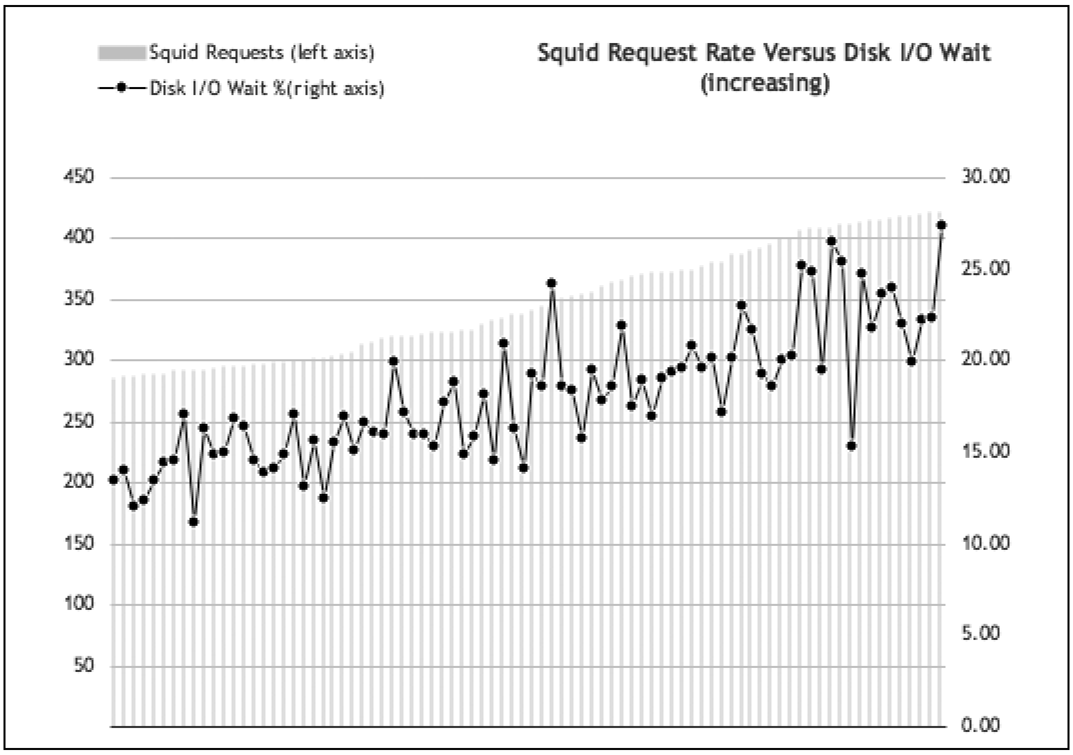


Figure 3-18. Squid request rate versus disk I/O wait: increasing view

Squid keeps internal metrics regarding the time it takes to handle both cache hits and misses. We can collect those metrics in Ganglia as well. We're not very concerned about the time spent for cache misses, because for the most part that doesn't inform us as to the upper limits of our cache —handling a miss mostly strains the network and origin server. Hits come from Squid's memory and disk cache, so that's where we want to focus our attention. Figure 3-19 presents the results over a five-month period.

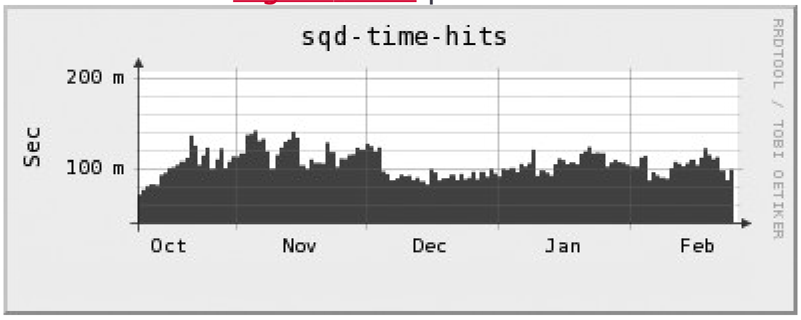


Figure 3-19. Squid hit time (in milliseconds): five-month view

This illustrates how the time for a cache hit has not changed significantly over the time period we're measuring (hovering around 100 milliseconds). Note that squid's "time-to-serve" metrics include the time until the client finishes receiving the last byte of the response, which can vary depending on how far your clients are from the server. This informs us that while disk I/O wait has increased, it has not affected the response time of the Squid server—at least not with the load it has been experiencing. We'd like to keep the time-to-serve metric within a reasonable range so the user isn't forced to wait for photos, so we’ve arbitrarily set a maximum time-to-serve of 180 milliseconds for this particular server; we’ll still want to stay below that. But what amount of traffic *will* push our time-to-serve above that threshold?

In order to find that out, let’s go to our stalwart load-increaser exercise. We’ll want to increase production load slowly on the servers while recording their metrics. Since we now know which of our hardware resources follows increasing traffic, we know what to watch for: the threshold at which disk I/O wait starts to affect cache hit response time.

Increasing the request rate to our Squid server should be done slowly to avoid completely flooding our hit ratio. As depicted in Figure 3-20, by either replaying URLs via Httperf or Siege, or by removing servers from a load-balanced pool, we can bump up the request rate gradually on a single Squid server.

As you can see, the service time increases along with the disk I/O wait time (no surprise there). Due to a wide range of photo sizes, there’s a lot of variation in our data for time-to-serve, but we begin to see 180 millisecond serving times at approximately 40 percent disk I/O wait. The only task remaining is to find the request rate at which we hit that threshold (see Figure 3-21).

Here, we see the “red line” we’ve been looking for (metaphorically speaking). At 40 percent disk I/O wait, we’re processing upwards of 850 requests per second. If we use time-to-serve as a guide, this is going to be the maximum performance we can expect from our hardware platform with this particular configuration. As a point of reference, that configuration comprises a Dell PowerEdge 2950 with six 15,000 RPM SAS drives, 4 GB of RAM, and a single quad-core CPU.

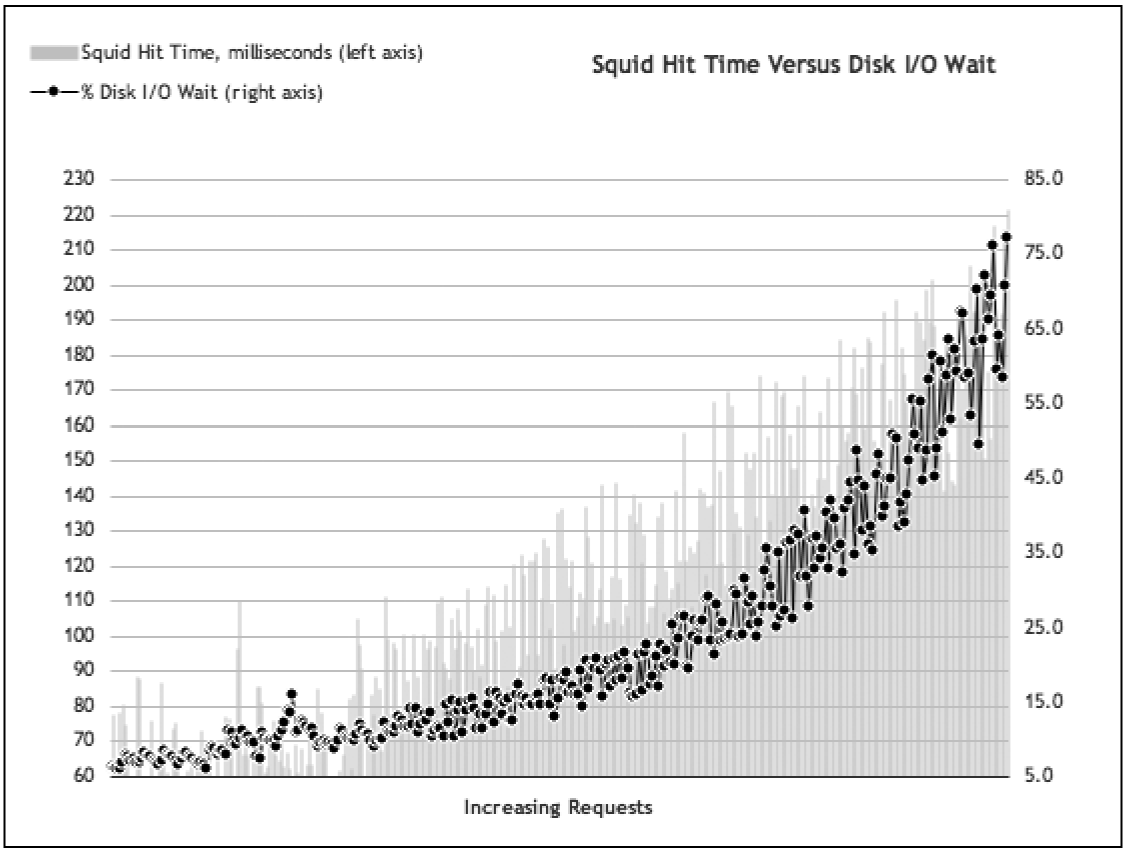


Figure 3-20. Testing for Squid ceilings: serving time versus disk I/O wait

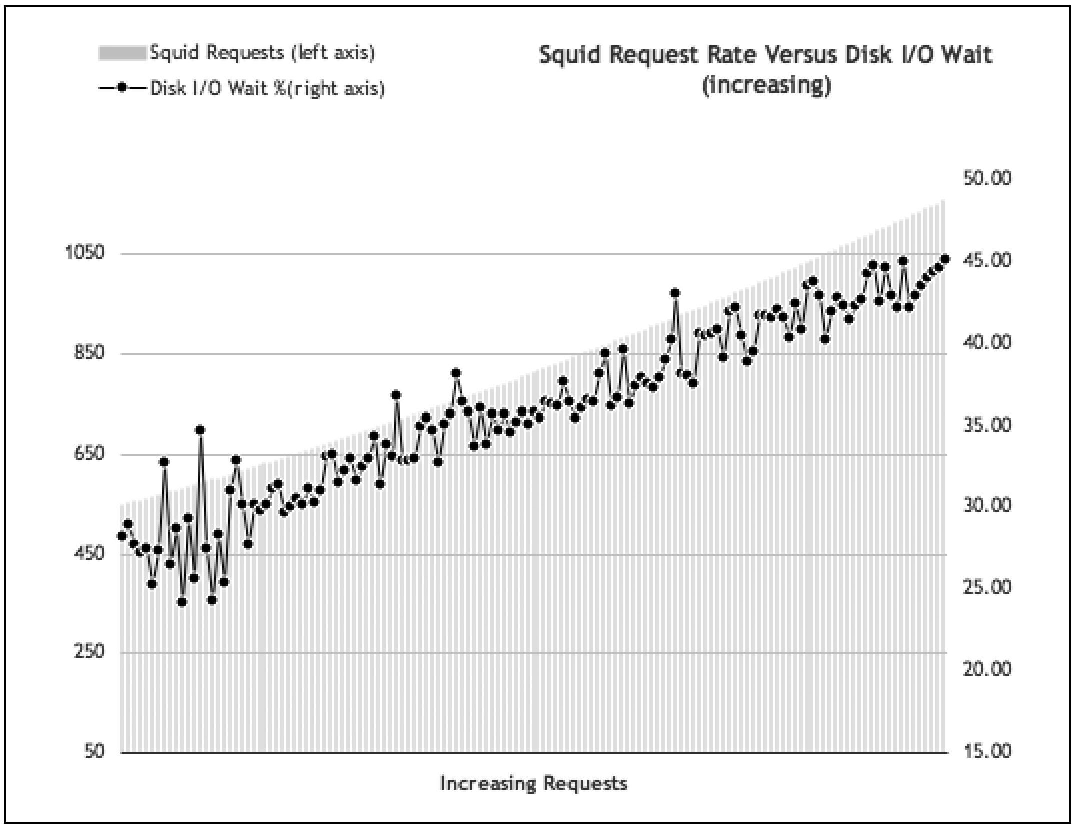
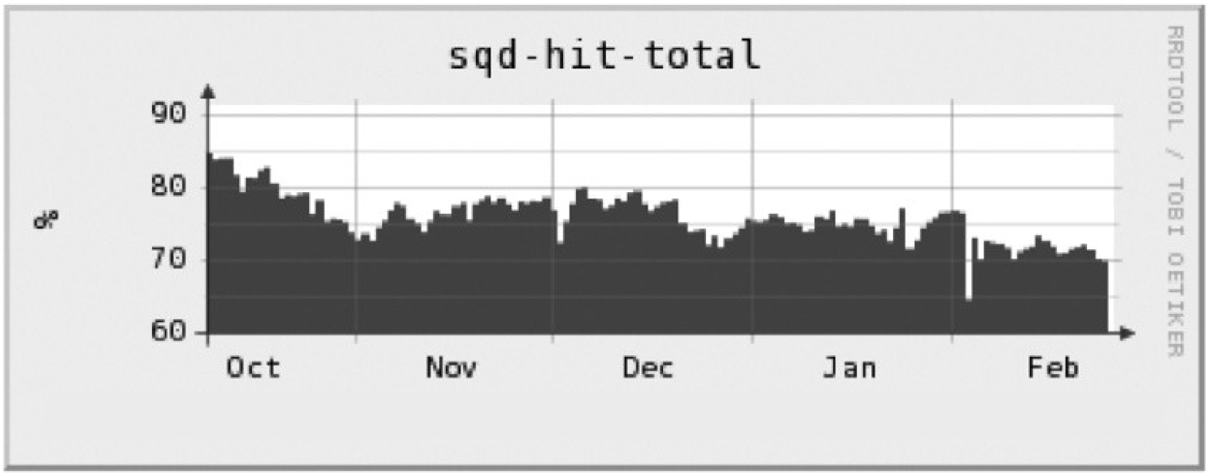


Figure 3-21. Squid request rate versus disk I/O wait, ceiling

However, we’re not done with our cache measurements. We also need to confirm how our cache’s efficiency changes over time, as we’re handling a dynamic working set. Figure 3-22 presents the results from a five-month view.



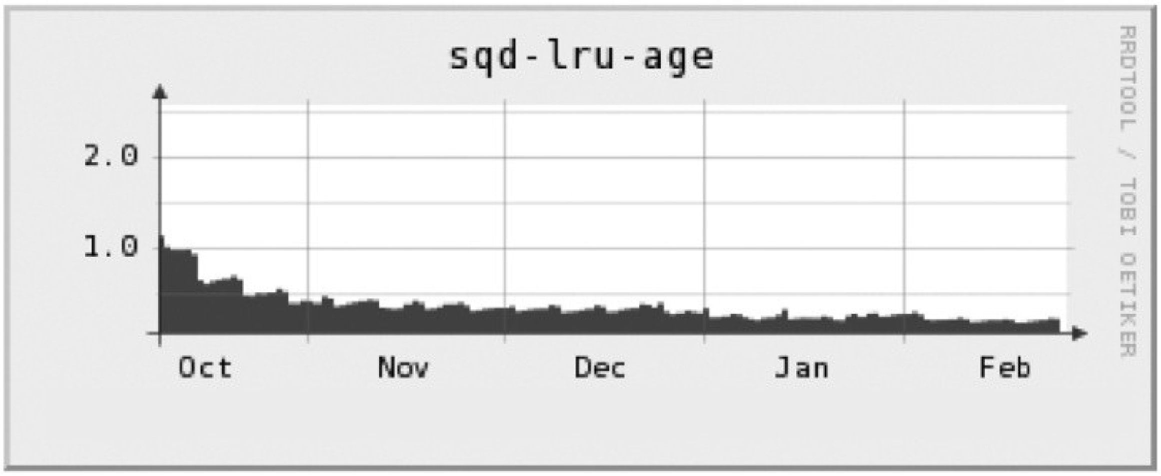


Figure 3-22. Cache hit ratios (%) and LRU age (days): five-month view

These two graphs display the hit ratio and LRU reference age of a particular Squid caching server we used to serve images over a period of five months. The hit ratio is expressed as a percentage, and the LRU reference age as units of days. During that time, LRU age and hit rate have both declined at a small but discernible rate, which we can attribute to the increase in photos being uploaded by users. As the working set of requested photos grows, the cache needs to work increasingly harder to evict objects to make room for new ones.

But even with this decrease in efficiency, it appears with a 72 percent hit rate, our LRU reference age for this server is about three hours. This is nothing to sneeze at, and is perfectly acceptable for our environment. We’ll want to keep an eye on the hit rate as time goes on and continue to tune the cache size as appropriate.

To summarize, this exercise involved two metrics related to the ceilings experience by our caching system: disk I/O wait and cache efficiency.

As the request rate goes up, so does the demand on our disk subsystem and our time-to-serve. At roughly 850 requests per second, we can maintain what we deem to be an acceptable end-user experience. As we approach that number, we'll want to increase the number of cache servers to comfortably handle the load.

The ceiling of 850 requests per second assumes we have a steady cache hit ratio, which could also change over time.

Special Use and Multiple Use Servers

In our web server example, CPU usage was our defining metric. Admittedly, this makes the job pretty easy; you have a fixed amount of CPU to work with. It was also made less difficult by virtue of the fact Apache was the only significant application using the CPU. There are many circumstances though, in which you don't have the luxury of dedicating each server to do a single task. Having a server perform more than one task—email, web, uploads—can make more efficient use of your hardware, but it complicates taking measurements.

Our process thus far has been to tie system resources (CPU, memory, network, disk, and so on) to application-level metrics (Apache hits, database queries, etc.). When you run many different processes, it's difficult to track how their usage relates to each other, and how each process might signal it's reaching the upper limits of efficient operation.

But simply because this scenario can complicate capacity measurements, you need not assume it makes planning impossible.

In order to discern what processes are consuming which resources, you'll want to do one of the following:

- Isolate each running application and measure its resource consumption
- Hold some of the applications' resource usage constant in order to measure one at a time

The process requires some poking around in the data to find situations in which events just happen to run controlled experiments for you. For instance, as we'll see in the example later in this section, I happened to notice two days had similar web server traffic but different CPU usage. I could exploit this oddity to find out the constraints on capacity for the web server.

Once upon a time at Flickr, the photo upload and processing tasks resided on the same machines that were serving pages for the Flickr.com website; that configuration made capacity planning difficult. Image processing is a highly CPU-intensive task, and as the number of uploads increased, so did the dependence on disk I/O resources. Add to that the increase in our traffic, and we quickly discovered three different jobs were all fighting for the same resources.

At any given moment, we weren't exactly certain how much hardware was being used for each process, so we added some application-level metrics to guide our estimates:

- Photo uploads (which mostly drove disk I/O and network utilization)
- Image processing (which drove CPU utilization)
- Serving the site's pages (which drove both memory and CPU utilization)

I already knew our traffic pattern and shape for each of our system metrics, and now I could correlate them to the tasks they were doing. I wanted to isolate the resource effects of each of those tasks to track them separately, or at least get a good idea of what was doing what. Having these metrics already stored in RRD files, I could dump their values to text then load them into Excel, where I could conveniently graph them.

First I found a two-day period in which the pattern of web traffic was the similar for both days (Figure 3-23). At that time, the burden on our web servers included not only serving the site's page, but taking in photo uploads and processing them as well.

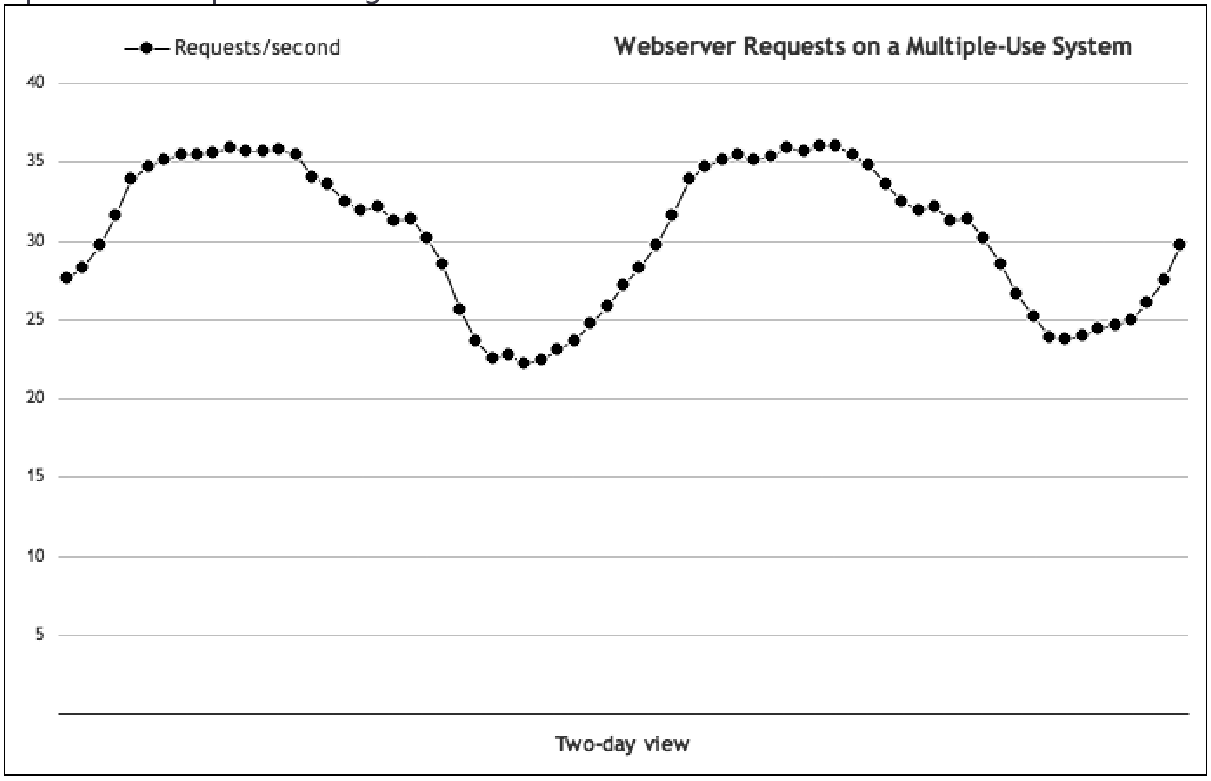


Figure 3-23. Two-day view of Apache requests

Now let’s take a look at our system resources. In Figure 3-24, we have overlaid the CPU usage from Figure 3-23 onto the web server data for the same time period.

Clearly, the CPU was working harder on the second day, even though the amount of Apache traffic was roughly the same. The only other task this server was doing was image processing, so we know that we can attribute the different CPU usage to that image processing. What we want to do next is quantify that effect (see Figure 3-25).

This graph uncovers what we suspected: the extra CPU consumption on the second day was due to photo processing. This activity actually occurred over a weekend, and as we mentioned earlier in the chapter, Sundays are a high upload day.

Figure 3-26 plots the photo processing rates for both days against each other and shows the differences. Note that while the peaks for Sunday were more than 20 percent higher, during the evening the rate dropped below that of Saturday’s rates at the same time, making the difference negative on the graph.

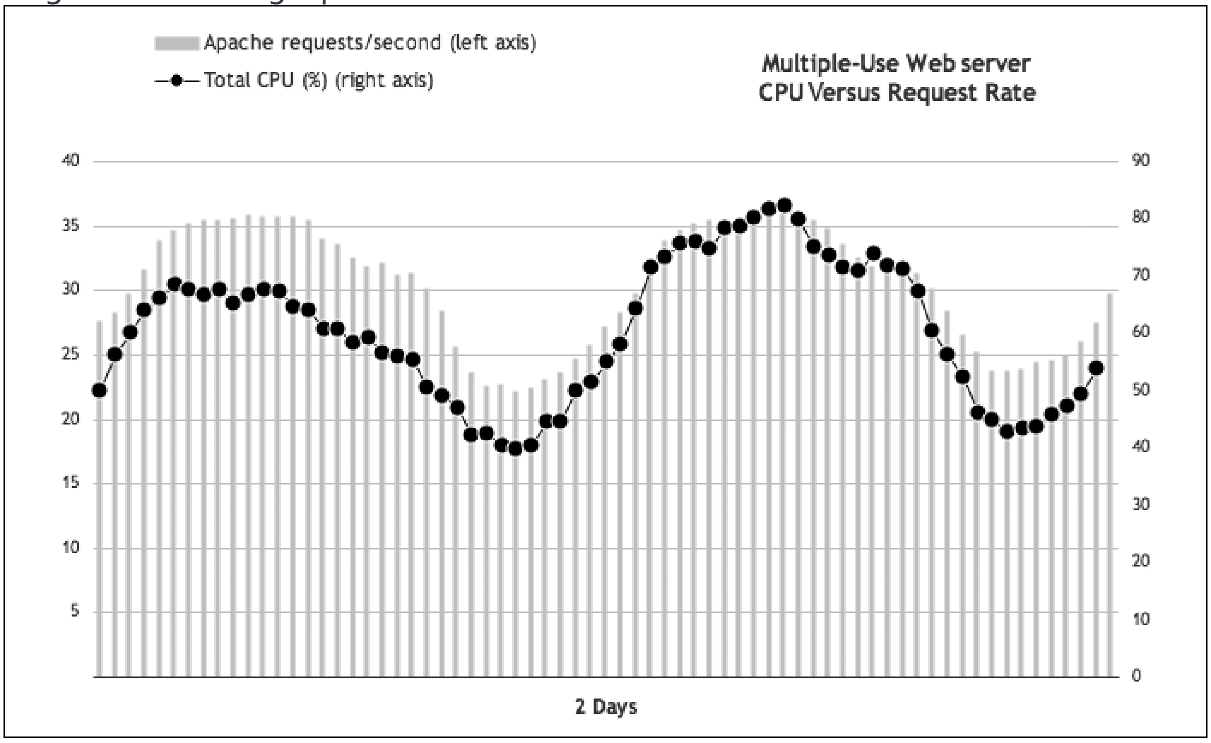


Figure 3-24. Web server work versus total CPU: two-day view

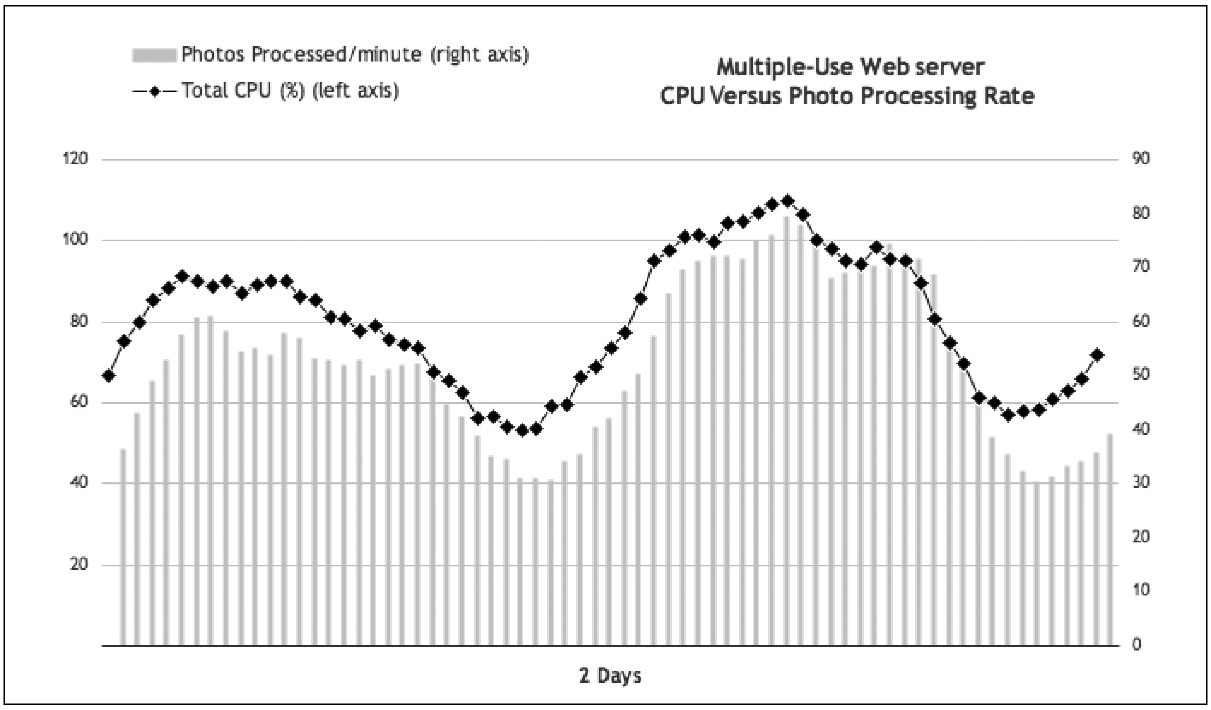


Figure 3-25. Total CPU versus photo processing rate, two-day view

We had all the data we needed to make an educated guess as to how much CPU is required to process images (as opposed to the amount required to serve Apache requests). We just needed to mine the data to come up with the specific values (Figure 3-27).

Figure 3-27 points out, at least for this given weekend, every 30 photos processed per minute can equate to an additional 25 percent CPU utilization.

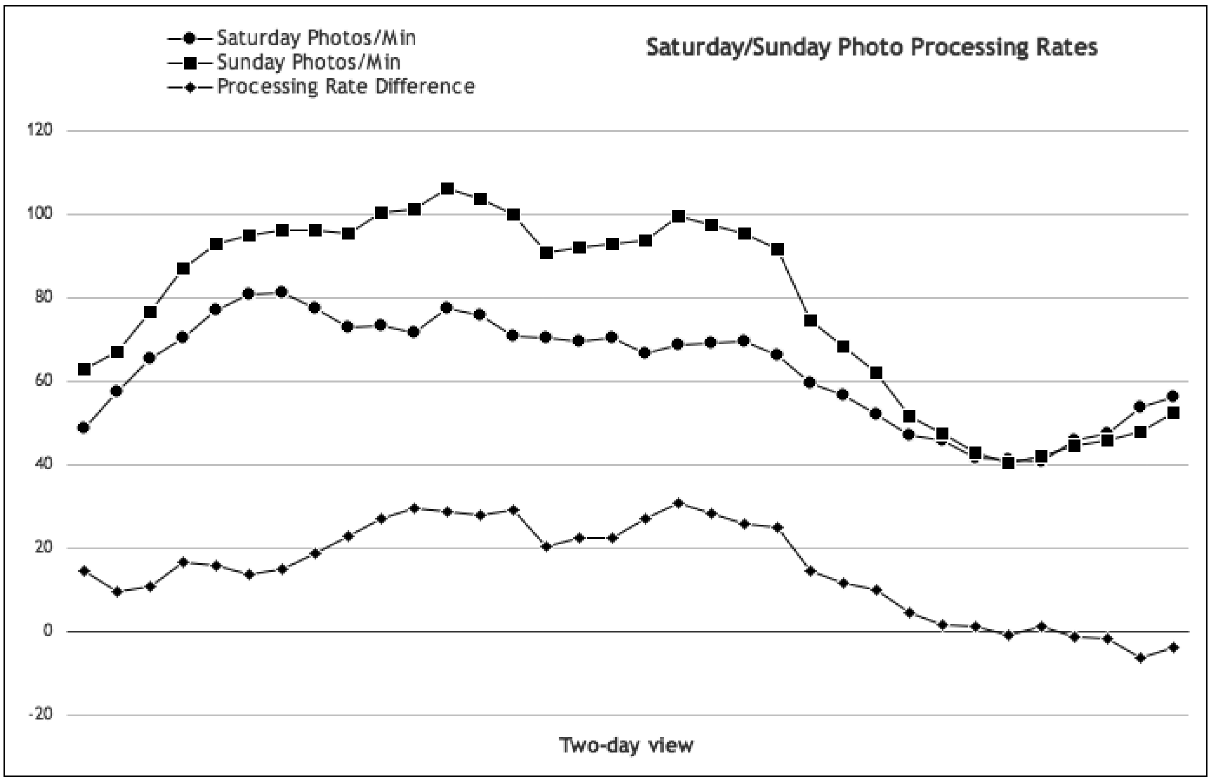


Figure 3-26. Saturday and Sunday photo processing rates

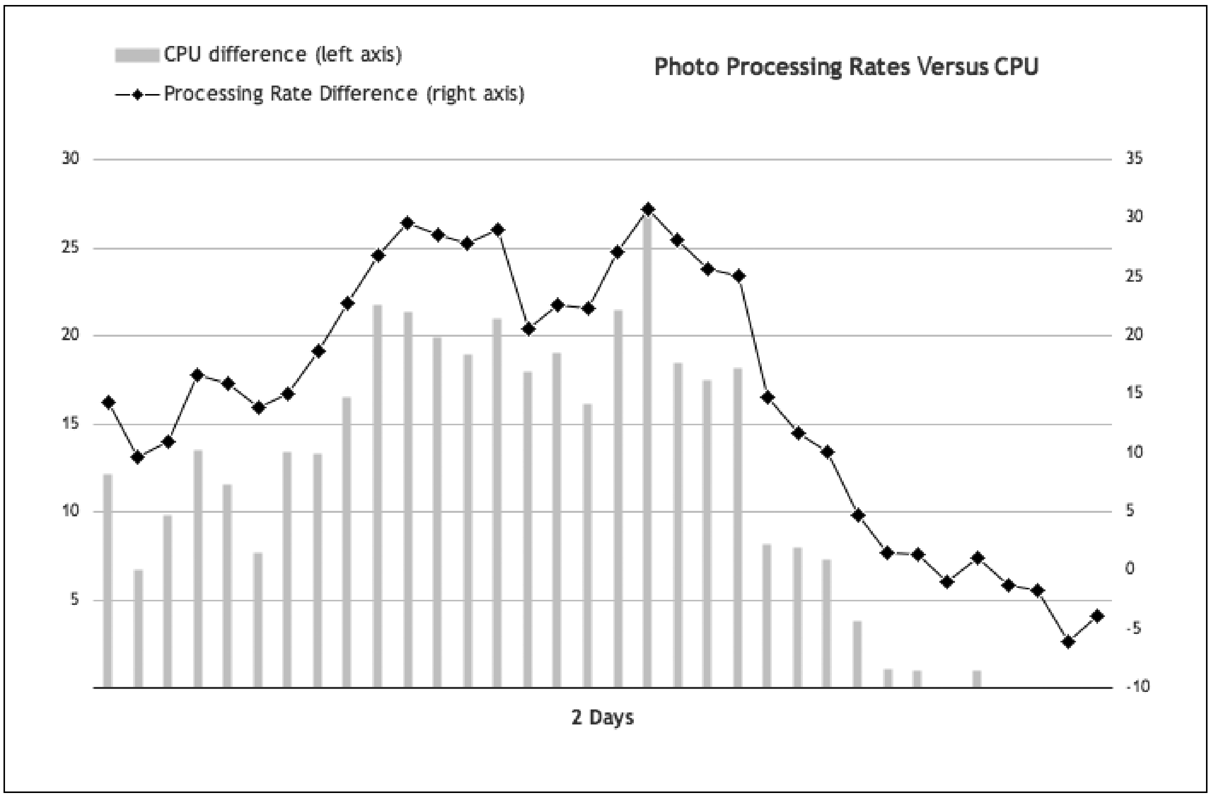


Figure 3-27. Photo processing rate and CPU usage, distilled from two-day data

This is an extremely rough estimate based on a small and statistically insignificant set of data, and you should consider this nothing more than an example of how to isolate resource usage on a multiple-use system. Confirming this 25:30 ratio using proper rigor would mean looking at a larger range of traffic and upload rates, and comparing the data again. But this process gives us a starting point from which we can base our ceilings.

In this situation, an ideal scenario is one in which we're tracking the two variables (web traffic and upload rates) and can figure out how many machines we'll need if we continue running both processes on each machine. This process worked for over a year in planning the capacity for Flickr's web servers. We finally split out the image processing onto its own dedicated cluster that could take advantage of multicore CPUs—another example of diagonal scaling at work.

API Usage and Its Effect on Capacity

As more and more websites use open APIs to open up their services to external developers, capacity planning for the use of those services must follow.

You may have guessed by now I’m a strong advocate of application-level metrics as well as system metrics, and API usage is the area where application-level metrics can matter the most. When you allow others access to your data via an open API, you’re essentially allowing a much more focused and routine use of your website.

One of the advantages of having an open API is it allows for more efficient use of your application. If external developers wanted to gain access to your data and no API methods existed, they might *screen scrape* your site’s pages to get at the data, which is extremely inefficient for a number of reasons. If they’re only interested in a specific piece of data on the page, they’d still have to request the entire page and everything that entails, such as downloading CSS markup, JavaScript, and other components necessary for a client’s browser to render the page, but of no interest to the developer. Although APIs allow more efficient use of your application, if not tracked properly, they also expose your web service to potential abuse, as they enable other applications to ask for those specific pieces of data.

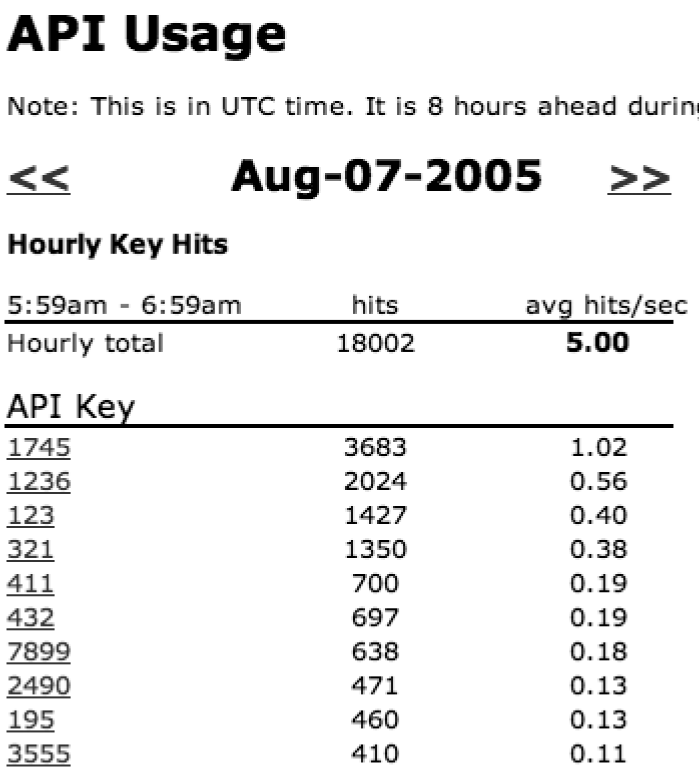
Having some way to measure and record the usage of your open API on a per-user, or per-request-method basis, should be considered mandatory in capacity tracking on a site offering a web API. This is commonly done through the use of unique API *keys,* or other unique credentials. Upon each call to the API, the key identifies the application and the developer responsible for building the application.

Because it’s much easier to issue an enormous volume of calls to an API than to use a regular client browser, you should keep track of what API calls are being made by what application, and at what rate.

At Flickr, we automatically invalidate any key that appears to be abusing the API, according to provisions outlined in the Terms of Service. We maintain a running total every hour for every API key that makes a call, how many calls were made, and the details of each call. See Figure 3-28 for the basic idea of API call metrics.

With this information, you can identify which API keys are responsible for the most traffic. You should be keeping track of the details of each key, as shown in Figure 3-29.

By collecting this information on a regular basis, you’ll have a much better idea of how API usage affects your resources. You can then adjust API limits as your capacity landscape changes.


API Usage

Note: This is in UTC time. It is 8 hours ahead during daylight savings.

<< Aug-07-2005 >>

Hourly Key Hits

5:59am - 6:59am	hits	avg hits/sec
Hourly total	18002	**5.00**

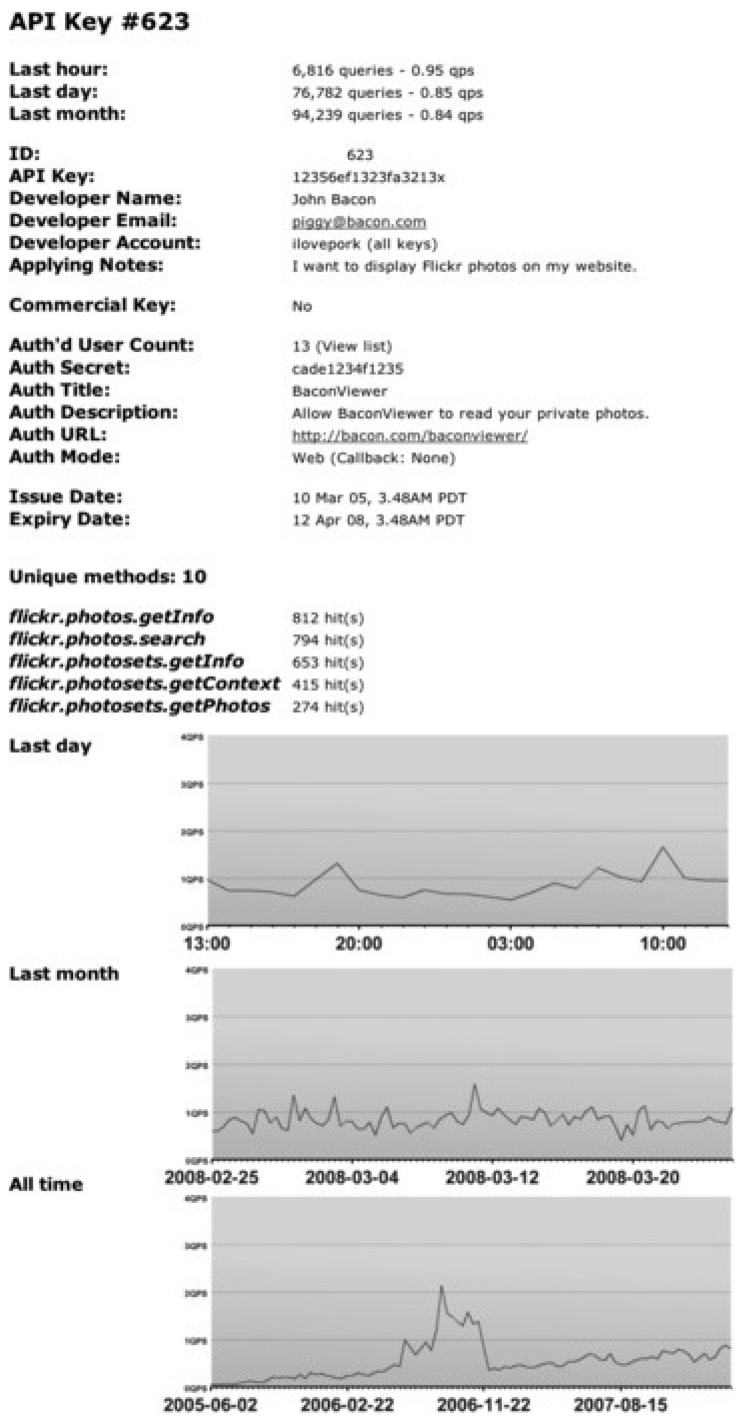
API Key		
1745	3683	1.02
1236	2024	0.56
123	1427	0.40
321	1350	0.38
411	700	0.19
432	697	0.19
7899	638	0.18
2490	471	0.13
195	460	0.13
3555	410	0.11

Figure 3-28. Keeping track of API request statistics

Examples and Reality

Will *your* web servers, databases, storage, and caches exhibit the same behavior as these? It's almost guaranteed they won't because each application and type of data affects system resources differently. The examples in this chapter simply illustrate the methods and thought processes by which you can investigate and form a better understanding of how increased load can affect each part of your infrastructure.
The important lesson to retain is each segment of your architecture will spend system resources to serve your website, and you should make sure you're measuring those resources appropriately. However, recording the right measurements isn't enough. You need to have some idea of when those resources will run out, and that's why you periodically need to probe to establish those ceilings.

Running through the exercise of finding your architecture's upper limits can reveal bottlenecks you didn't even know existed. As a result, you might make changes to your application, your hardware, your network, or any other component responsible for the problem. Every time you make a change to your architecture, you'll need to check ceilings again, because they're likely to change. This shouldn't be a surprise, because by now you know that capacity planning is a process, not a one-time event.


API Key #623

Last hour:	6,816 queries - 0.95 qps
Last day:	76,782 queries - 0.85 qps
Last month:	94,239 queries - 0.84 qps
ID:	623
API Key:	12356ef1323fa3213x
Developer Name:	John Bacon
Developer Email:	piggy@bacon.com
Developer Account:	ilovepork (all keys)
Applying Notes:	I want to display Flickr photos on my website.
Commercial Key:	No
Auth'd User Count:	13 (View list)
Auth Secret:	cade1234f1235
Auth Title:	BaconViewer
Auth Description:	Allow BaconViewer to read your private photos.
Auth URL:	http://bacon.com/baconviewer/
Auth Mode:	Web (Callback: None)
Issue Date:	10 Mar 05, 3.48AM PDT
Expiry Date:	12 Apr 08, 3.48AM PDT

Unique methods: 10

flickr.photos.getInfo	812 hit(s)
flickr.photos.search	794 hit(s)
flickr.photosets.getInfo	653 hit(s)
flickr.photosets.getContext	415 hit(s)
flickr.photosets.getPhotos	274 hit(s)

Last day

Last month

All time

Figure 3-29. API key details and history

Summary

Measurement is a necessity, not an option. It should be viewed as the eyes and ears of your infrastructure. It can inform all parts of your organization: finance, customer care, engineering, and product management.

Capacity planning can’t exist without the measurement and history of your system and application-level metrics. Planning is also ineffective without knowing your system's upper performance boundaries so you can avoid approaching them. Finding the ceilings of each part of your architecture involves the same process:

1. Measure and record the server's primary function.
   Examples: Apache hits, database queries
2. Measure and record the server's fundamental hardware resources.
   Examples: CPU, memory, disk, network usage
3. Determine how the server's primary function relates to its hardware resources.
   Examples: n database queries result in m percent CPU usage
4. Find the maximum acceptable resource usage (or ceiling) based on both the server's primary function and hardware resources by one of the following:
   - Artificially (and carefully) increasing real production load on the server through manipulated load balancing or application techniques.
   - Simulating as close as possible a real-world production load.

Chapter 4. Predicting Trends

I'M ASSUMING YOU'VE MADE A FEW PASSES THROUGH Chapter 3 AND HAVE JUST DEPLOYED A SUPER-AWESOME, totally amazing, monitoring, trending, graphing, and measurement system. You're graphing everything you can get your hands on, as often as you can. You probably didn't gain anything from graphing the peak barking periods of your neighbor's dog—but hey, you did it, and I'm proud of you.

Now you'll be able to use this data (excluding the barking statistics) like a crystal ball, and predict the future like Nostradamus. But let's stop here for a moment to remember an irritating little detail: *it's impossible to accurately predict the future*.

Forecasting capacity needs is part intuition, and part math. It's also the art of slicing and dicing up your historical data, and making educated guesses about the future. Outside of those rare bursts and spikes of load on your system, the long-term view is hopefully one of steadily increasing usage. By putting all of this historical data into perspective, you can generate estimates for what you'll need to sustain the growth of your website. As we'll see later, the key to making accurate predictions is having an *adjustable* forecasting process.

ORA: DON'T BUY BEFORE YOU NEED IT

Before you get too excited about charting massive growth and putting new servers in place to handle the deluge, let me remind you of one of the key economic factors you need to deal with: buying equipment too early is wasteful.

This rule is derived directly from the obvious trend in computing costs: all forms of hardware are becoming cheaper, even as they become faster and more reliable. Whether or not Moore's Law (Gordon E. Moore's now-famous axiom in 1965 that postulates the number of transistors on an integrated circuit approximately doubles every eighteen months) holds true forever, we can predict that manufacturers will continue to lower costs over time. If you can wait six months before buying a piece of equipment, you will likely end up with faster and less expensive equipment at that time.

Certainly, you don't want to be caught unprepared when growth takes place—this book is all about saving you from that career-threatening situation. Conversely, the company financial officers will not hold you in high regard either when you've purchased a lot of equipment that lay idle, only to see its price drop a few months later.

Riding Your Waves

A good capacity plan depends on knowing your needs for your most important resources, and how those needs change over time. Once you have gathered historical data on capacity, you can begin analyzing it with an eye toward recognizing any trends and recurring patterns.

For example, in the last chapter I recounted how at Flickr, we discovered Sunday has been historically the highest photo upload day of the week. This is interesting for many reasons. It may also lead us to other questions: has that Sunday peak changed over time, and if so, how has it changed with respect to the other days of the week? Has the highest upload day always been Sunday? Does that change as we add new members residing on the other side of the International Date Line? Is Sunday still the highest upload day on holiday weekends? These questions can all be answered once you have the data, and the answers in turn could provide a wealth of insight with respect to planning new feature launches, operational outages, or maintenance windows.

Recognizing trends is valuable for many reasons, not just for capacity planning. When we looked at disk space consumption in Chapter 3, we stumbled upon some weekly upload patterns. Being aware of any recurring patterns can be invaluable when making decisions later on. Trends can also inform community management, customer care and support, product management, and finance. Some examples of how metrics measurement can be useful include:

- Your operations group can avoid scheduling maintenance that could affect image processing machines on a Sunday, opting for a Friday instead, to minimize any adverse effects on users.
- If you deploy any new code that touches the upload processing infrastructure, you might want to pay particular attention the following Sunday to see whether everything is holding up well when the system experiences its highest load.
- Making customer support aware of these peak patterns allows them to gauge the effect of any user feedback regarding uploads.
- Product management might want to launch new features based on the low or high traffic periods of the day. A good practice is to make sure everyone on your team knows where these metrics are located and what they mean.
- Your finance department might also want to know about these trends because it can help them plan for capital expenditure costs.

Trends, Curves, and Time

Let's take a look back at the daily storage consumption data we collected in the last chapter and apply it to make a forecast of future storage needs. We already know the defining metric: total available disk space. Graphing the cumulative total of this data provides the right perspective from which to predict future needs. Taking a look at Figure 4-1, we can see where we’re headed with consumption, how it’s changing over time, and when we’re likely to run out of space.

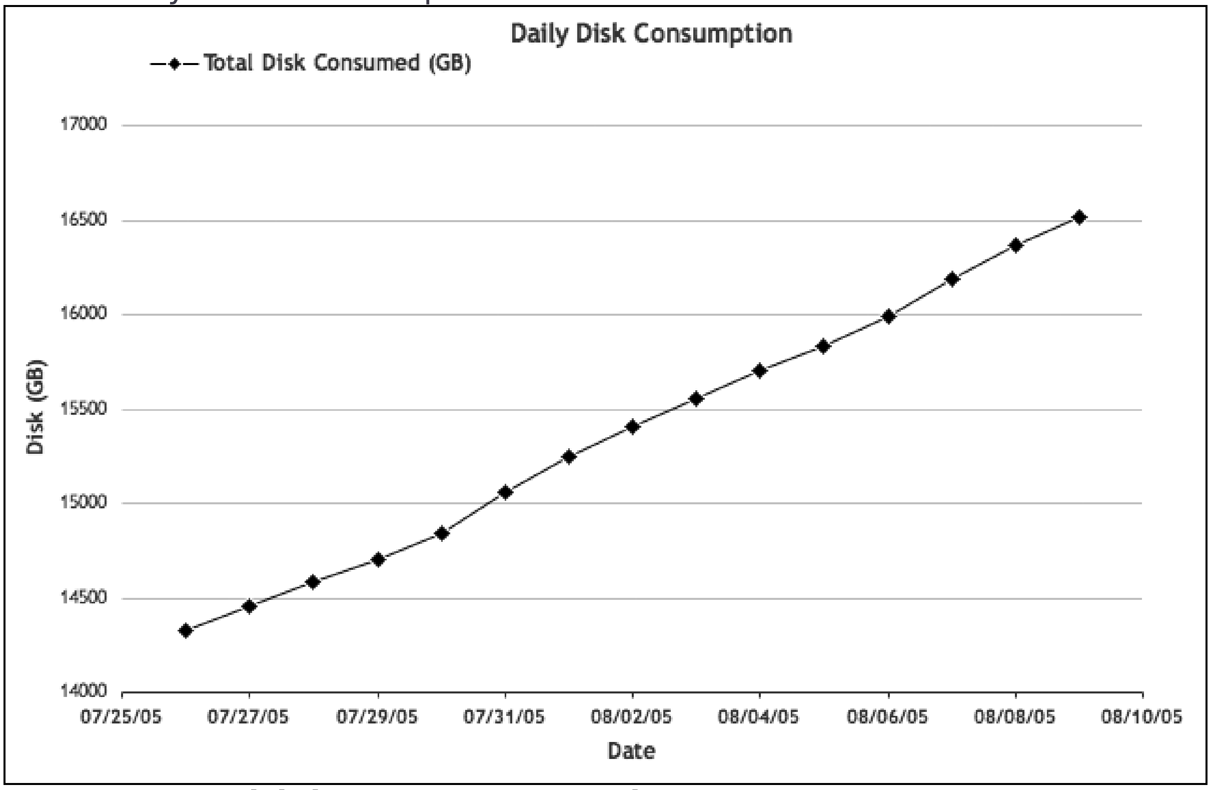


Figure 4-1. Total disk consumption: cumulative view

Now, let’s add our constraint: the total currently available disk space. Let’s assume for this example we have a total of 20 TB (or 20,480 GB) installed capacity. From the graph, we see we’ve consumed about 16 TB. Adding a solid line extending into the future to represent the total space we have installed, we obtain a graph that looks like Figure 4-2. This illustration demonstrates a fundamental principal of capacity planning: predictions require two essential bits of information, your ceilings and your historical data.

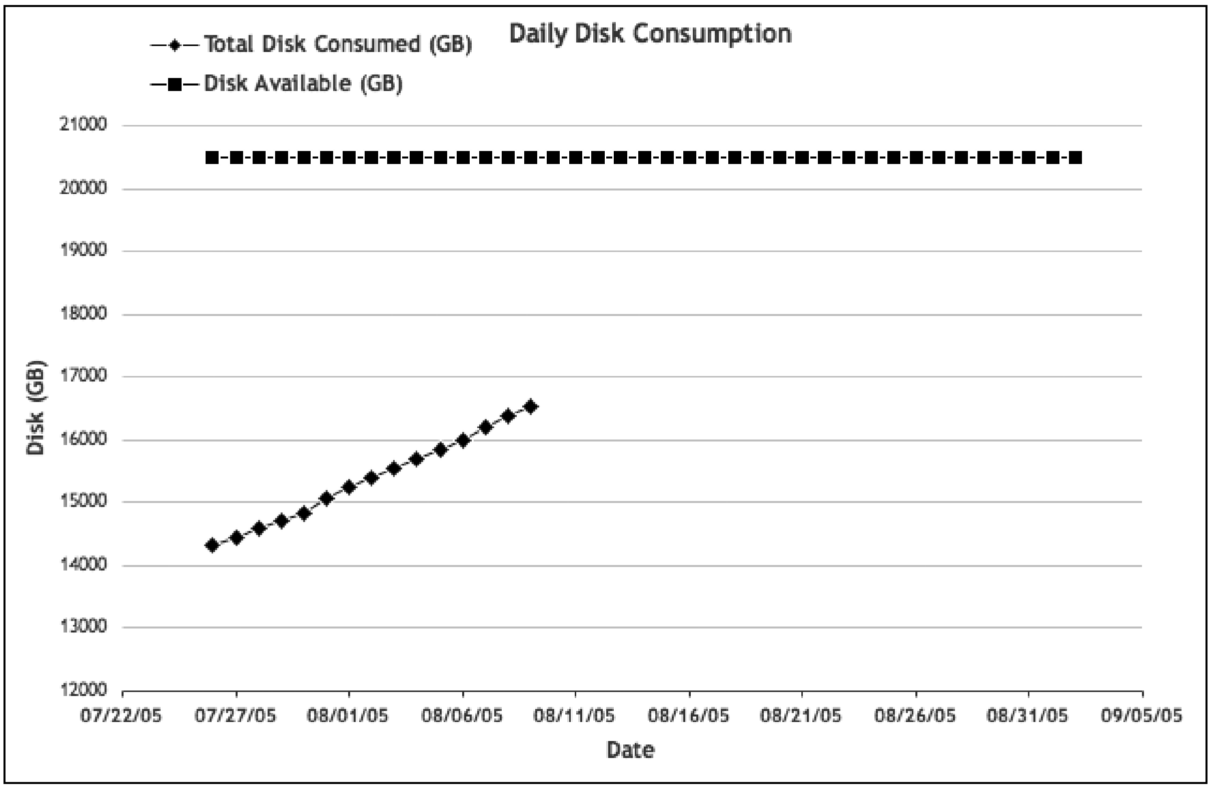
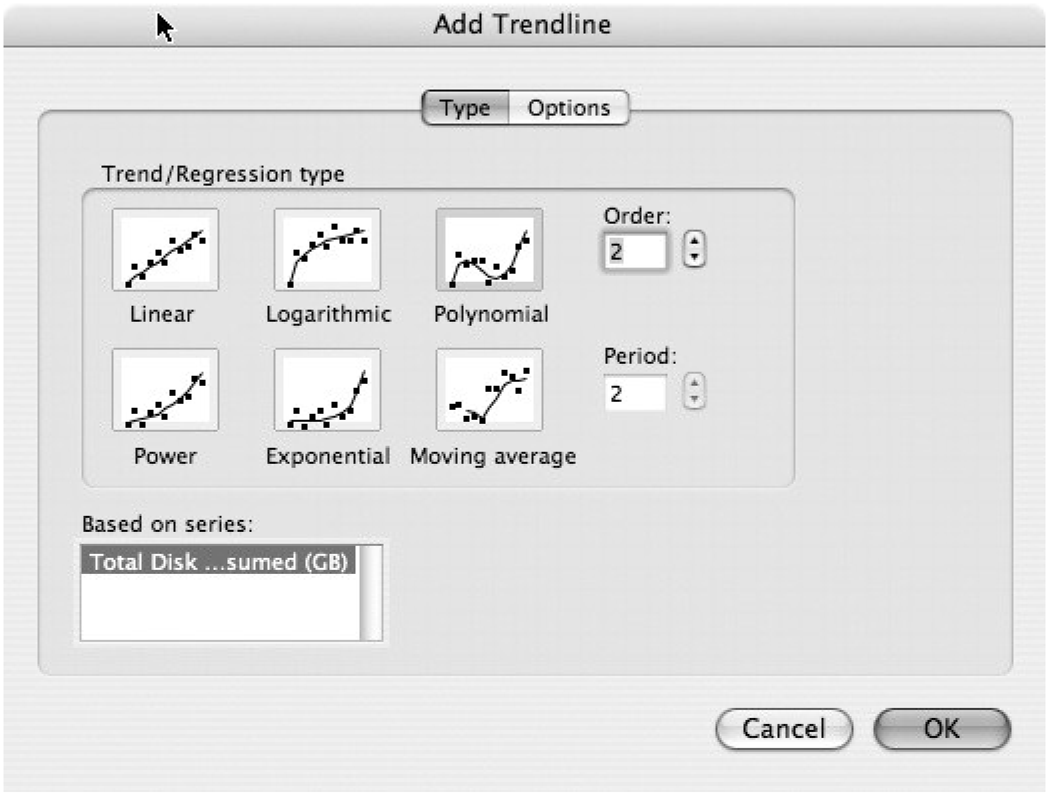


Figure 4-2. Cumulative disk consumption and available space

Determining when we're going to reach our space limitation is our next step. As I just suggested, we could simply draw a straight line that extends from our measured data to the point at which it intersects our current limit line. But is our growth actually linear? It may not be.

Excel calls this next step "adding a trend line," but some readers might know this process as *curve fitting*. This is the process by which you attempt to find a mathematical equation that mimics the data you're looking at. You can then use that equation to make educated guesses about missing values within the data. In this case, since our data is on a time line, the missing values in which we're interested are in the future. Finding a good equation to fit the data can be just as much art as science. Fortunately, Excel is one of many programs that feature curve fitting.
To display the trend using a more mathematical appearance, let's change the Chart Type in Excel from Line to XY (Scatter).

XY (Scatter) changes the date values to just single data points. We can then use the trending feature of Excel to show us how this trend looks at some point in the future. Right-click the data on the graph to display a drop-down menu. From that menu, select Add Trendline. A dialog box will open, as shown in Figure 4-3.



Order: 2

Period: 2

Based on series: Total Disk ...sumed (GB)

Figure 4-3. Add Trendline Type dialog box

Next, select a trend line type. For the time being, let's choose Polynomial, and set Order to 2. There may be good reasons to choose another trend type, depending on how variable your data is, how much data you have, and how far into the future you want to extrapolate. For more information, see the upcoming sidebar, "Fitting Curves."

In this example, the data appears about as linear as can be, but since I already know this data isn't linear over a longer period of time (it's accelerating), I'll pick a trend type that can capture some of the acceleration we know will occur.

After selecting a trend type, click the Options tab to bring up the Add Trendline options dialog box, as shown in Figure 4-4.
To show the equation that will be used to mimic our disk space data, click the checkbox for "Display equation on chart." We can also look at the R^2 value for this equation by clicking the "Display R-squared value on chart" checkbox.
The R^2 value is known in the world of statistics as the *coefficient of determination*. Without going into the details of how this is calculated, it's basically an indicator of how well an equation matches a certain set of data. An R^2 value of 1 indicates a mathematically perfect fit. With the data we're using for this example, any value above 0.85 should be sufficient. The important thing to know is, as your R^2 value decreases, so too should your confidence in the forecasts. Changing the trend type in the previous step affects the R^2 values—sometimes for better, sometimes for worse—so

some experimentation is needed here when looking at different sets of data.

ORA: FITTING CURVES

In capacity planning, curve fitting is where the creative can collide with the scientific. In most cases, capacity planning is used to stay ahead of growth, which is generally represented through time-series data that extends upward and to the right in some form or manner.

Figuring out how and when the data gets there is the challenge, and we aim to use *extrapolation* to solve it. Extrapolation is the process of constructing new data points beyond a set of known data points. In our case, we're going to be defining new data points that exist in the future of time-series data.

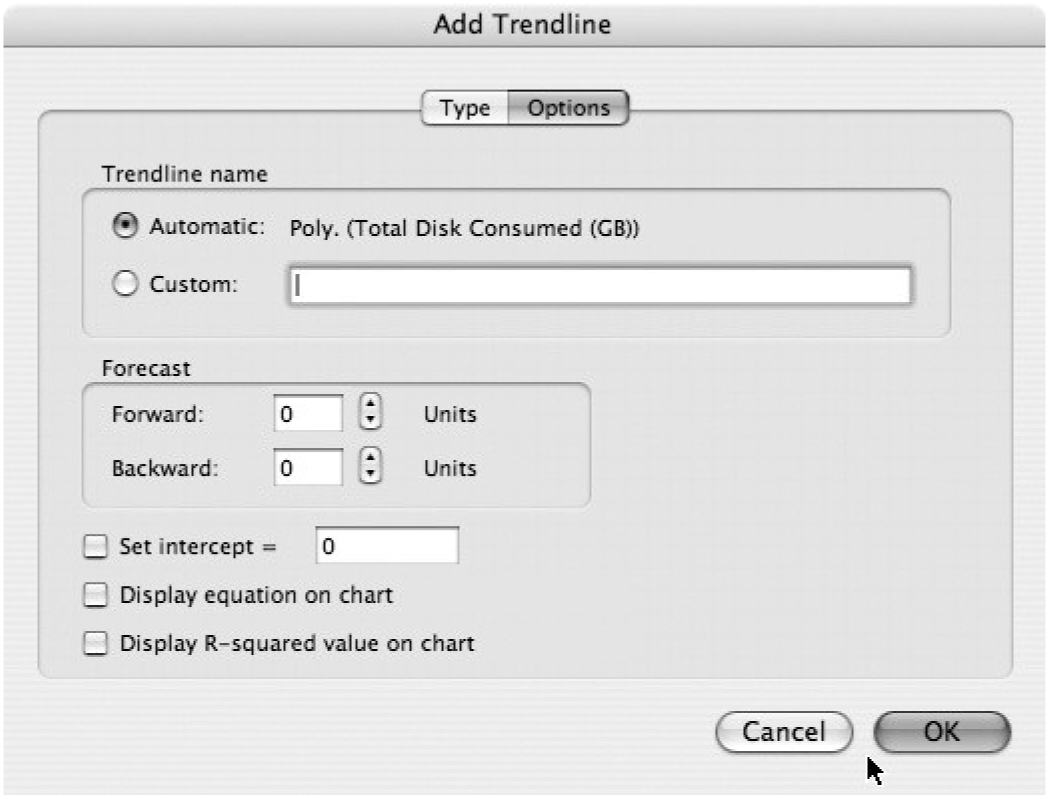
The difficulty with curve fitting and extrapolation is you need to reconcile what you know about the source of your data with the apparent best-fit equation. Simply because you find a curve that fits the data with 99.999% mathematical accuracy doesn't mean it's going to be an accurate picture of the future. Your data will almost always have context outside of the mathematical equation. For example, forecasting the sale of snow shovels must include considerations for time of year (winter versus summer) or geography (Alaska or Arizona).

When finding equations to fit your data, it's also best to stay away from higher-order polynomial equations. They're tempting because their fit (*coefficient of determination*) is so good, but anything higher than a 2nd order polynomial can exhibit dramatic fluctuations outside of the dataset you're looking at.

The moral of the story is to use a good deal of common sense when curve-fitting your data. Don't insist on elegantly perfect fits, as they are quite often the result of questionable assumptions.

We'll want to extend our trend line into the future, of course. We want to extend it far enough into the future such that it intersects the line corresponding to our total available space. This is the point at which we can predict we'll run out of space. Under the Forecast portion of the dialog box, enter 25 units for a value. Our units in this case are days. After you hit OK, you'll see our forecast looks similar to Figure 4-5.

The graph indicates that somewhere around day 37, we run out of disk space. Luckily, we don't need to squint at the graph to see the actual values; we have the equation used to plot that trend line. As detailed in Table 4-1, plugging the equation into Excel, and using the day units for the values of X, we find the last day we're below our disk space limit is 8/30/05.


Add Trendline

Type | Options

Trendline name

[✓ Automatic:] Poly. (Total Disk Consumed (GB))

Custom:

Forecast

Forward: 0 Units

Backward: 0 Units

☐ Set intercept = 0

☐ Display equation on chart

☐ Display R-squared value on chart

Figure 4-4. Add Trendline Options dialog box

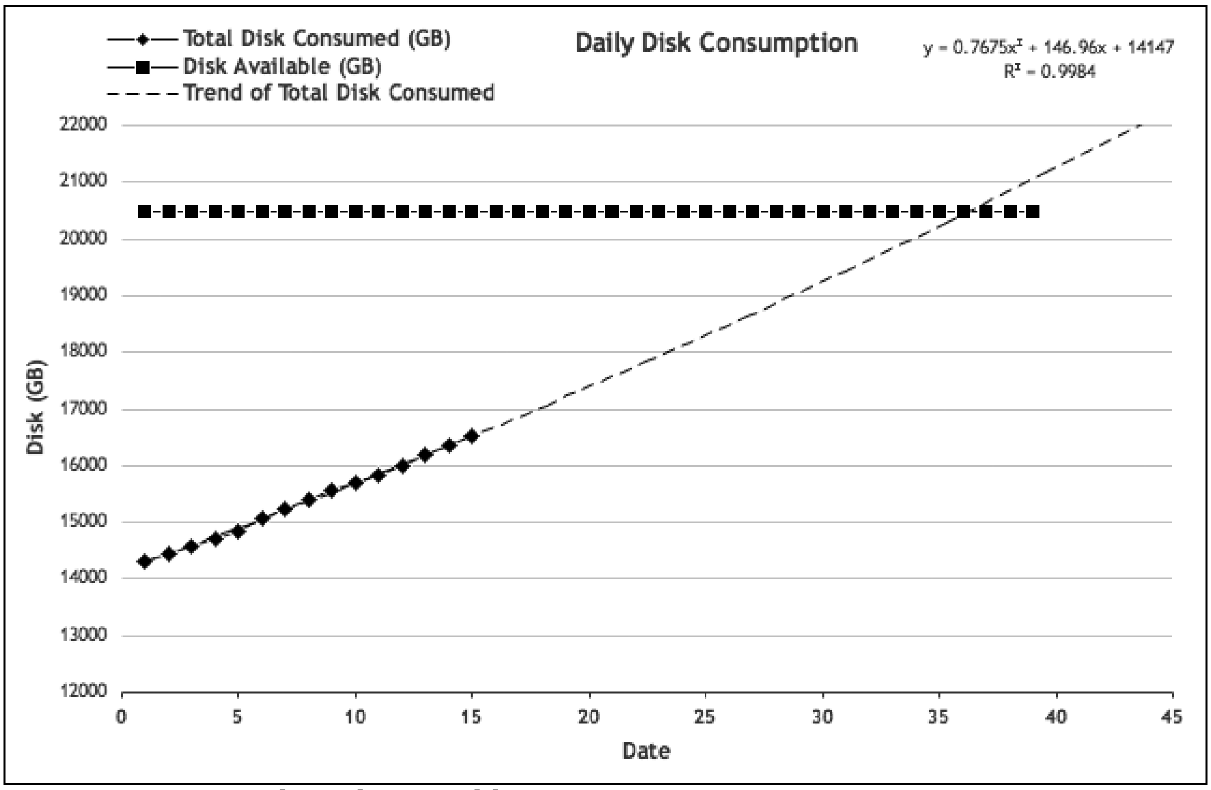


Figure 4-5. Extending the trend line

Table 4-1. Determining the precise day you will run out of disk space

	Date	Disk available (GB)	y=0.7675 x 2 + 146.96x + 14147
33	08/27/05	20480.00	19832.49
34	08/28/05	20480.00	20030.87
35	08/29/05	20480.00	20230.79
36	08/30/05	20480.00	20432.24
37	08/31/05	20480.00	20635.23
38	09/01/05	20480.00	20839.75
39	09/02/05	20480.00	21045.81

Now we know when we'll need more disk space, and we can get on with ordering and deploying it.

This example of increasing disk space is about as simple as they come. But as the metric is consumption-driven, every day has a new value that contributes to the definition of our curve. We also need to factor in the peak-driven metrics that drive our capacity needs in other parts of our site. Peak-driven metrics involve resources that are continually regenerated, such as CPU time and network bandwidth. They fluctuate more dramatically and thus are more difficult to predict, so curve fitting requires more care.

Tying Application Level Metrics to System Statistics: Database Example

In Chapter 3, we went through the exercise of establishing our database ceiling values. We discovered (through observing our system metrics) that 40 percent disk I/O wait was a critical value to avoid, because it's the threshold at which database replication begins experiencing disruptive lags.

How do we know when we'll reach this threshold? We need some indication when we are approaching our ceiling. It appears the graphs don't show a clear and smooth line just bumping over the 40 percent threshold. Instead, our disk I/O wait graph shows our database doing fine until a 40 percent spike occurs. We might deem occasional (and recoverable) spikes to be acceptable, but we need to track how our average values change over time so the spikes aren't so close to our ceiling. We also need to somehow tie I/O wait times to our database usage, and ultimately, what that means in terms of actual application usage.

To establish some control over this unruly data, let's take a step back from the system statistics and look at the purpose this database is actually serving. In this example, we're looking at a *user* database. This is a server in our main database cluster, wherein a segment of Flickr users store the metadata associated with their user account: their photos, their tags, the groups they belong to, and more. The two main drivers of load on the databases are, of course, the number of photos and the number of users.
This particular database has roughly 256,000 users and 23 million photos. Over time, we realized that neither the number of users nor the number of photos is singularly responsible for how much work the database does. Taking only one of those variables into account meant ignoring the effect of the other. Indeed, there may be many users who have few, or no photos; queries for their data is quite fast and not at all taxing. On the flip side, there are a handful of users who maintain enormous collections of photos.

We can look at our metrics for clues on our critical values. We have all our system metrics, our application metrics, and the historical growth of each.

We then set out to find the single most important metric that can define the ceiling for each database server. After looking at the disk I/O wait metric for each one, we were unable to distinguish a good correlation between I/O wait and the number of users on the database. We had some servers with over 450,000 users that were seeing healthy, but not dangerous, levels of I/O wait. Meanwhile, other servers with only 300,000 users were experiencing much higher levels of I/O wait. Looking at the number of photos wasn't helpful either—disk I/O wait didn't appear to be tied to photo population.

As it turns out, the metric that directly indicates disk I/O wait is the ratio of *photos-to-users* on each of the databases.
As part of our application-level dashboard, we measure on a daily basis (collected each night) how many users are stored on each database along with the number of photos associated with each user. The photos-to-user ratio is simply the total number of photos divided by the number of users. While this could be thought of as an average photos *per* user, the range

can be quite large, with some “power” Flickr users having many thousands of photos while a majority have only tens or hundreds. By looking at how the *peak* disk I/O wait changes with respect to this photos per user ratio, we can get an idea of what sort of application-level metrics can be used to predict and control the use of our capacity (see Figure 4-6).

This graph was compiled from a number of our databases, and displays the peak disk I/O wait values against their current photos-to-user ratios. With this graph, we can ascertain where disk I/O wait begins to jump up. There’s an elbow in our data around the 85–90 ratio when the amount of disk I/O wait jumps above the 30 percent range. Since our ceiling value is 40 percent, we’ll want to ensure we keep our photos-to-user ratio in the 80–100 range. We can control this ratio within our application by distributing photos for high-volume users across many databases.

I want to stop here for a moment to talk a bit about Flickr’s database architecture. After reaching the limits of the more traditional Master/Slaves MySQL replication architecture (in which all writes go to the master and all reads go to the slaves), we redesigned our database layout to be *federated*, or *sharded*. This evolution in architecture is becoming increasingly common as site growth reaches higher levels of changing data. I won’t go into how that architectural migration came about, but it’s a good example of how architecture decisions can have a positive effect on capacity planning and deployment. By federating our data across many servers, we limit our growth only by the amount of hardware we can deploy, not by the limits imposed by any single machine.

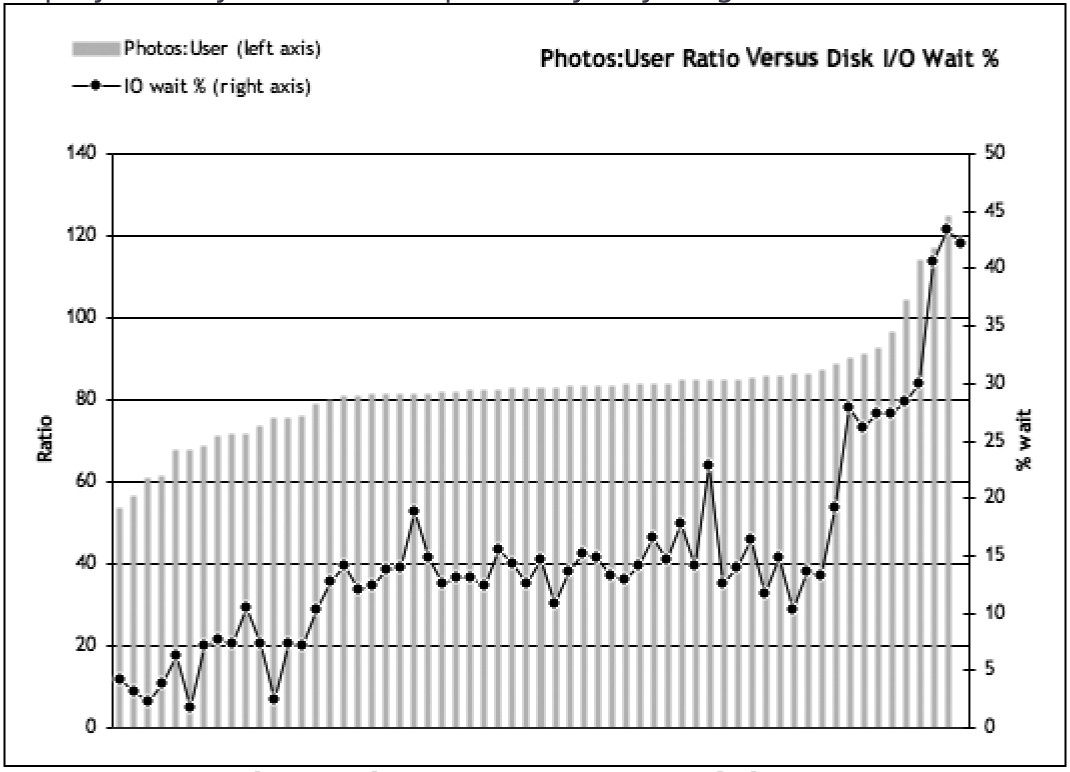


Figure 4-6. Database—photo:user ratio versus disk I/O wait percent

Because we're federated, we can control how users (and their photos) are spread across many databases. This essentially means each server (or pair of servers, for redundancy) contains a unique set of data. This is in contrast to the more traditional monolithic database that contains every record on a single server. More information about federated database architectures can be found in Cal Henderson's book, *Building Scalable Web Sites* (O'Reilly).

OK, enough diversions—let's get back to our database capacity example and summarize where we are to this point. Database replication lag is bad and we want to avoid it. We hit replication lag when we see 40 percent disk I/O wait, and we reach that threshold when we've installed enough users and photos to produce a photos-to-user ratio of 110. We know how our photo uploads and user registrations grow, because we capture that on a daily basis (Figure 4-7). We are now armed with all the information we need to make informed decisions regarding how much database hardware to buy, and when.

We can extrapolate a trend based on this data to predict how many users and photos we'll have on Flickr for the foreseeable future, then use that to gauge how our photos/user ratio will look on our databases, and whether we need to adjust the maximum amounts of users and photos to ensure an even balance across those databases.

We've found where the elbow in our performance (Figure 4-6) exists for these databases—and therefore our capacity—but what is so special about this photos/users ratio for our databases? Why does this particular value trigger performance degradation? It could be for many reasons, such as specific hardware configurations, or the types of queries that result from having that much data during peak traffic. Investigating the answers to these questions could be a worthwhile exercise, but here again I'll emphasize that we should simply expect this effect will continue and not count on any potential future optimizations.

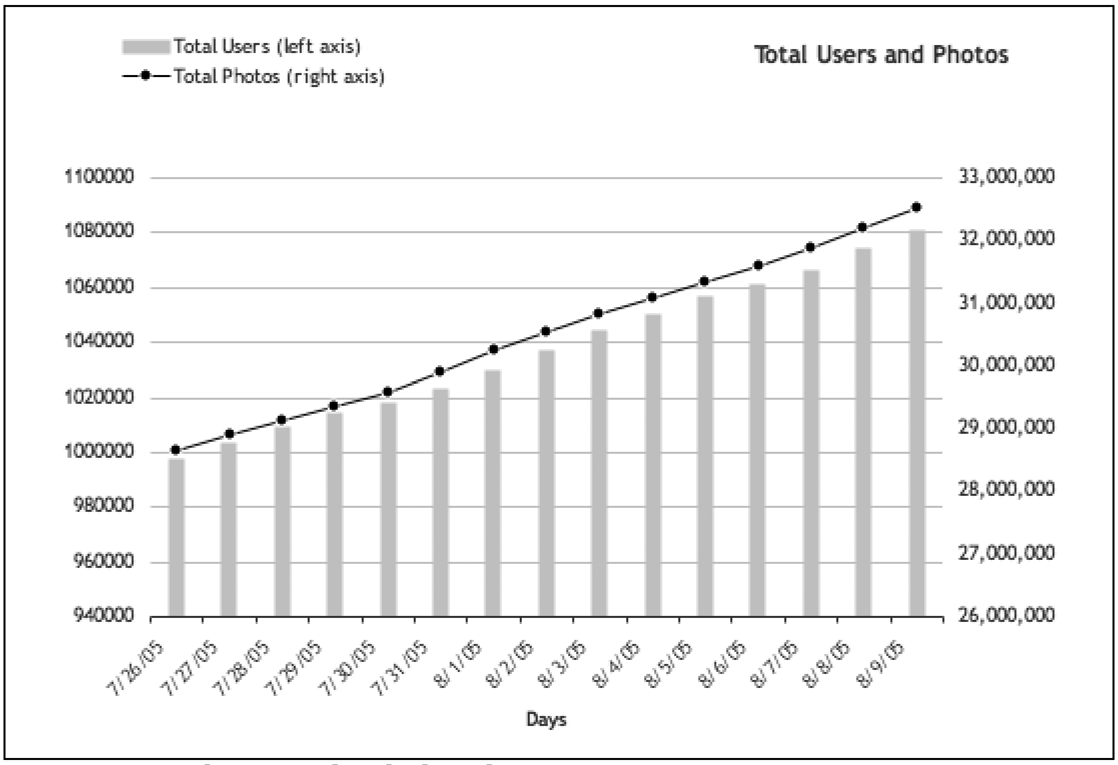


Figure 4-7. Photos uploaded and user registrations

Forecasting Peak-Driven Resource Usage: Web Server Example

When we forecast the capacity of a peak-driven resource, we need to track how the peaks change over time. From there, we can extrapolate from that data to predict future needs. Our web server example is a good opportunity to illustrate this process.

In Chapter 3, we identified our web server ceilings as 85 percent CPU usage for this particular hardware platform. We also confirmed CPU usage is directly correlated to the amount of work Apache is doing to serve web pages. Also as a result of our work in Chapter 3, we should be familiar with what a typical week looks like across Flickr's entire web server cluster. Figure 4-8 illustrates the peaks and valleys over the course of one week.
This data is extracted from a time in Flickr's history when we had 15 web servers. Let's suppose this data is taken today, and we have no idea how our activity will look in the future. We can assume the observations we made in the previous chapter are accurate with respect to how CPU usage and the number of busy apache processes relate—which turns out to be a simple multiplier: 1.1. If for some reason this assumption *does* change, we'll know quickly, as we're tracking these metrics on a per-minute basis. According to the graph in Figure 4-8, we're seeing about 900 busy concurrent Apache processes during peak periods, load balanced across 15 web servers. That works out to about 60 processes per web server. Thus, each web server is using approximately 66 percent total CPU (we can look at our CPU graphs to confirm this assumption).

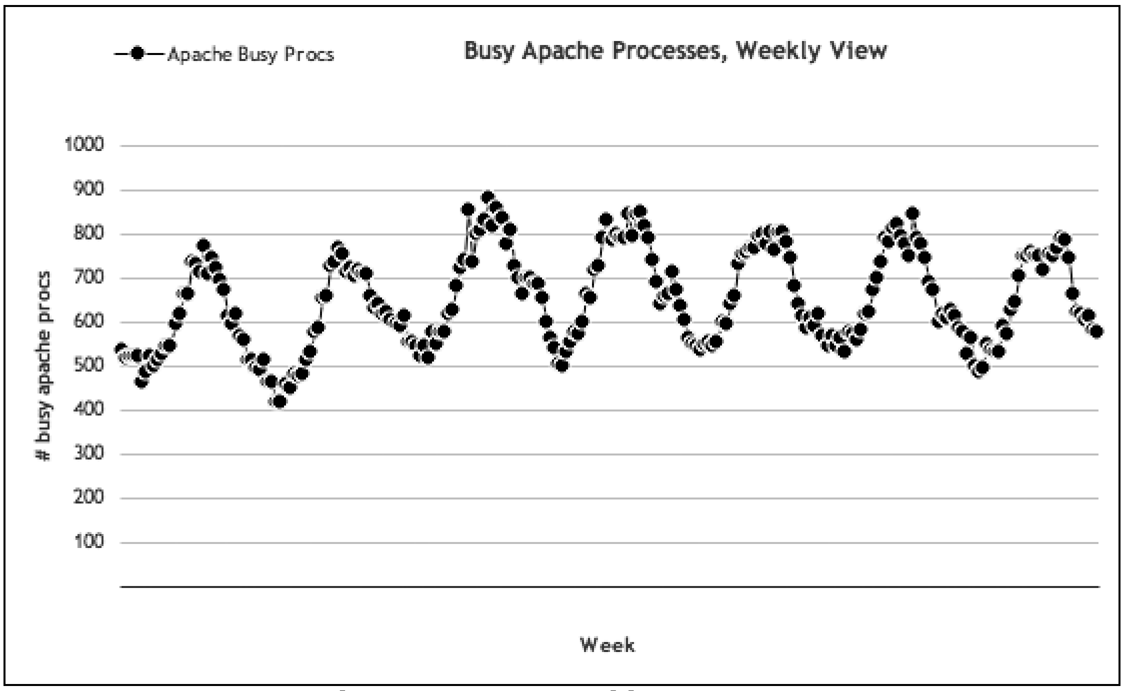


Figure 4-8. Busy Apache processes: weekly view

The peaks for this sample data are what we’re interested in the most. Figure 4-9 presents this data over a longer time frame, in which we see these patterns repeat.

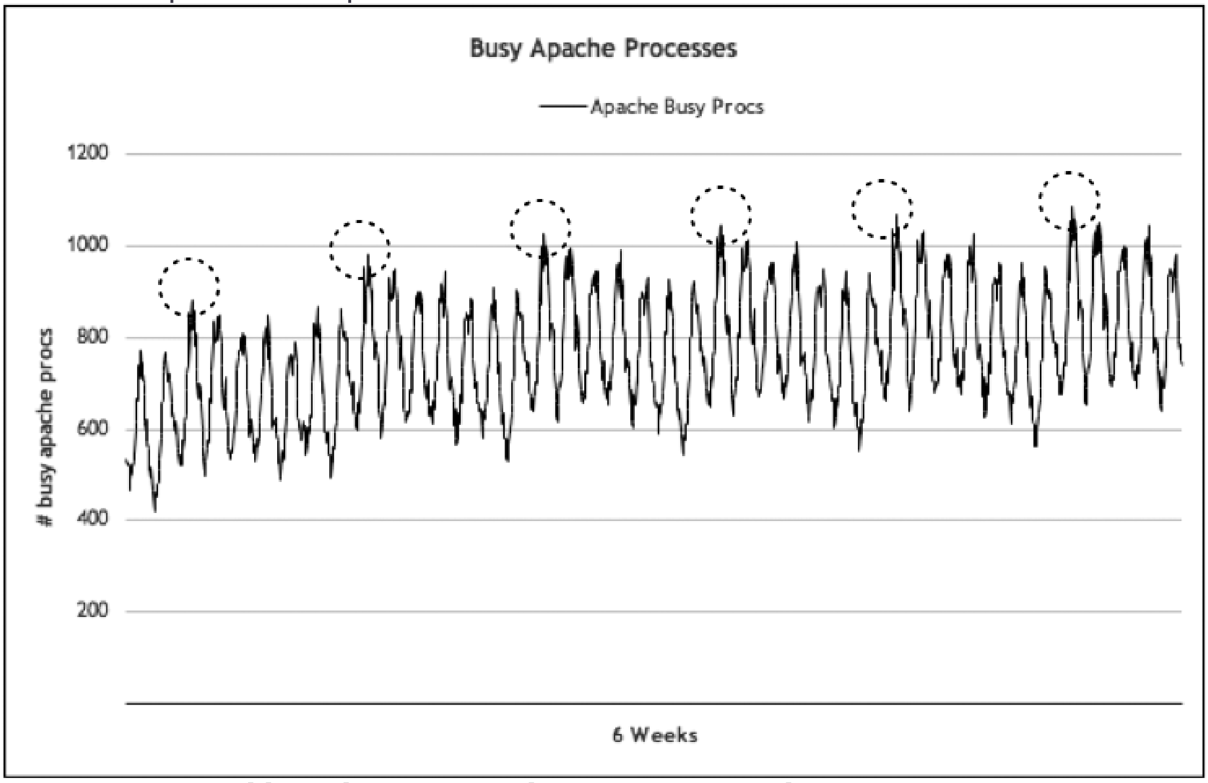


Figure 4-9. Weekly web server peaks across six weeks

It’s these weekly peaks that we want to track and use to predict our future needs. As it turns out, for Flickr, those weekly peaks almost always fall on a Monday. If we isolate those peak values and pull a trend line into the future as we did with our disk storage example above, we'll see something similar to Figure 4-10.

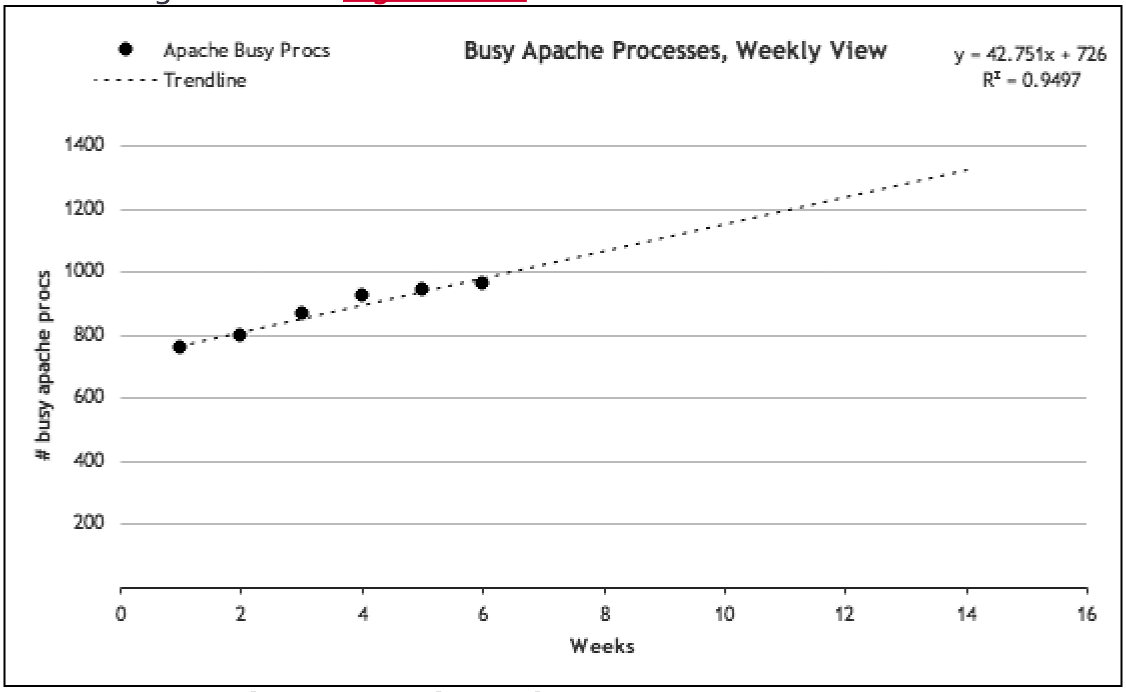


Figure 4-10. Web server peak trend

If our traffic continues to increase at the current pace, this graph predicts in another eight weeks, we can expect to experience roughly 1,300 busy Apache processes running at peak. With our 1.1 processes-to-CPU ratio, this translates to around 1,430 percent total CPU usage across our cluster. If we have defined 85 percent on each server as our upper limit, we would need 16.8 servers to handle the load. Of course, manufacturers are reluctant to sell servers in increments of tenths, so we'll round that up to 17 servers. We currently have 15 servers, so we'll need to add 2 more.

The next question is, when should we add them? As I explained in the sidebar "Don't Buy Before You Need It," we can waste a considerable amount of money if we add hardware too soon.

Fortunately, we already have enough data to calculate *when* we'll run out of web server capacity. We have 15 servers, each currently operating at 66 percent CPU usage at peak. Our upper limit on web servers is set at 85 percent, which would mean 1,275% CPU usage across the cluster. Applying our 1.1 multiplier factor, this in turn would mean 1,160 busy Apache processes at peak. If we trust the trend line shown in Figure 4-11, we can expect to run out of capacity sometime between the 9th and 10th week.
Therefore, the summary of our forecast can be presented succinctly:

- We'll run out of web server capacity three to four weeks from now.
- We'll need two more web servers to handle the load we expect to see in eight weeks.

Now we can begin our procurement process with detailed justifications based on hardware usage trends, not simply a wild guess. We'll want to ensure the new servers are in place before we need them, so we'll need to find out how long it will take to purchase, deliver, and install them.

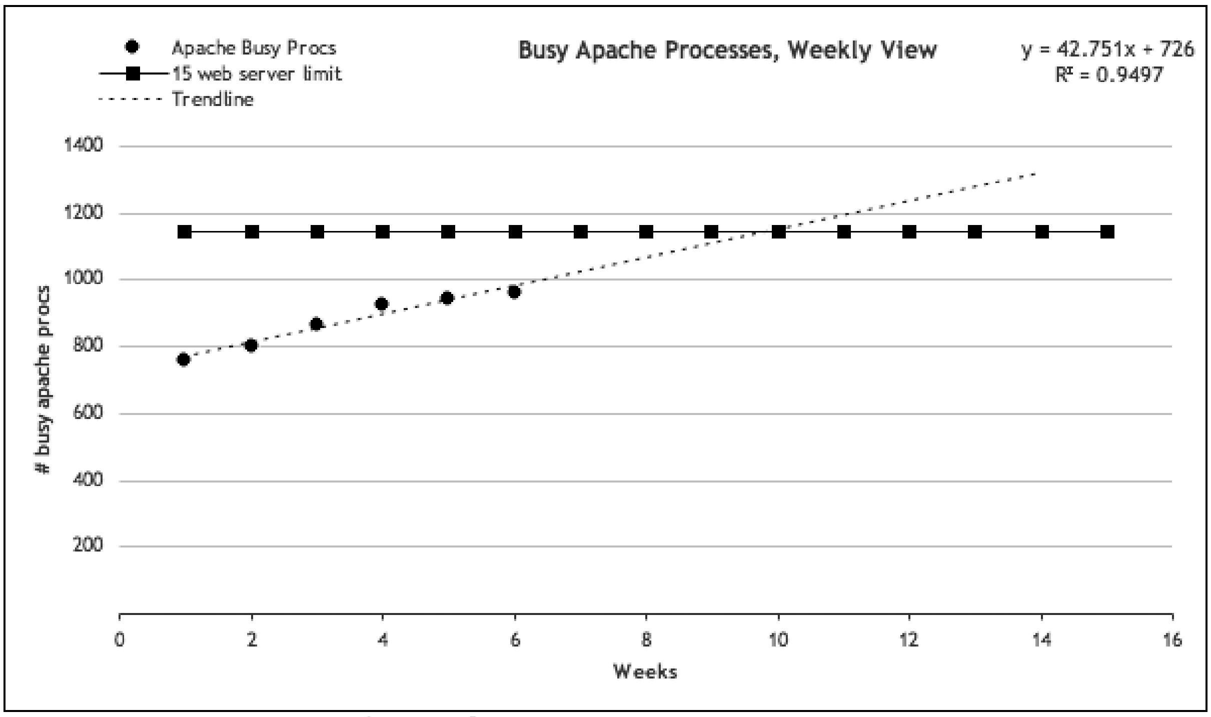


Figure 4-11. Capacity of 15 web servers

This is a simplified example. Adding two web servers in three to four weeks shouldn't be too difficult or stressful. Ideally, you should have more than six data points upon which to base your forecast, and you likely won't be so close to your cluster's ceiling as in our example. But no matter how much capacity you'll need to add, or how long the timeframe actually is, the process should be the same.

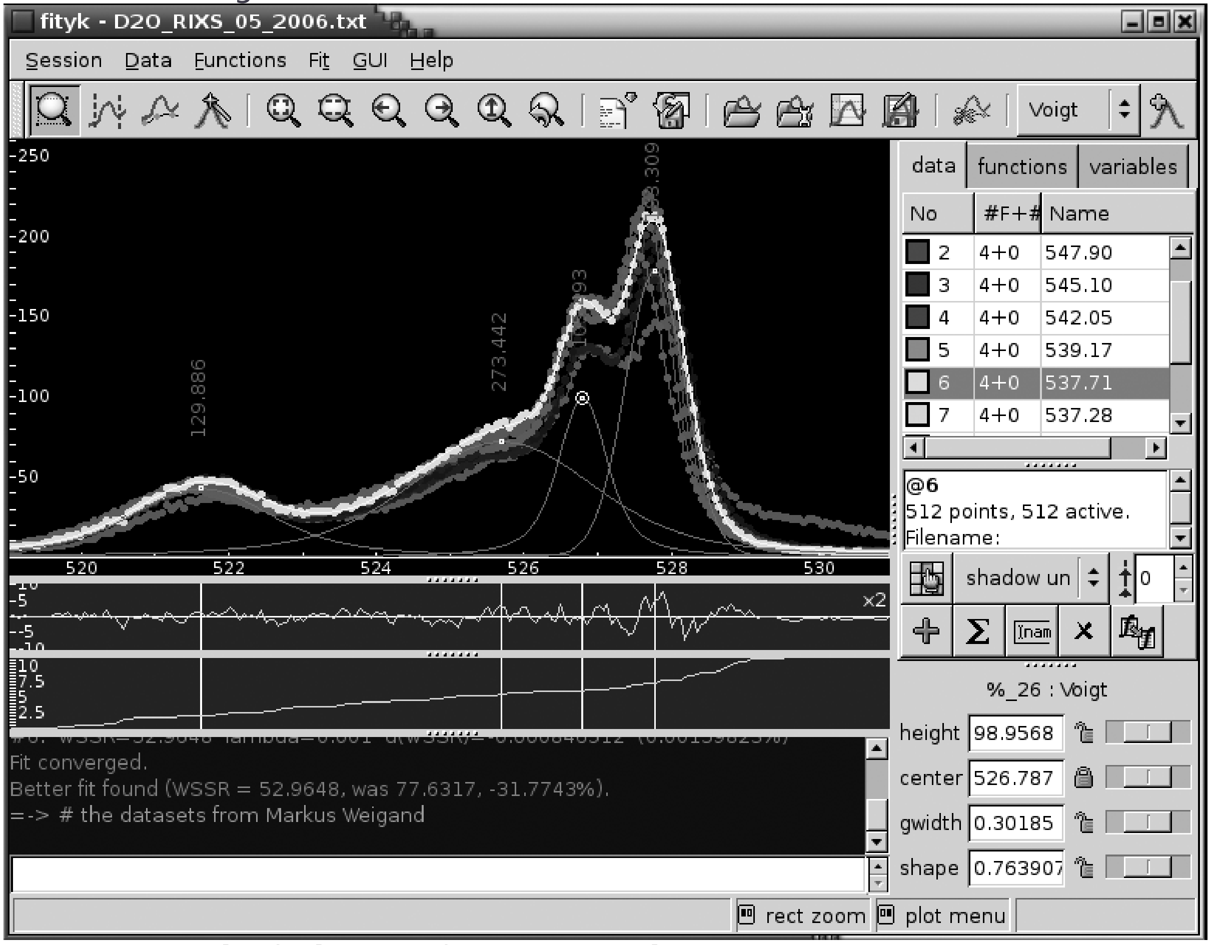
Caveats Concerning Small Data Sets

When you're forecasting with peak values as we've done, it's important to remember the more data you have to fit a curve, the more accurate your forecast will be. In our example, we based our hardware justifications on six weeks worth of data. Is that enough data to constitute a trend? Possibly, but the time period on which you're basing your forecasts is of great importance as well. Maybe there is a seasonal lull or peak in traffic, and you're on the cusp of one. Maybe you're about to launch a new feature that will add extra load to the web servers within the timeframe of this forecast. These are only a few considerations for which you may need to compensate when you're making justifications to buy new hardware. A lot of variables can come into play when predicting the future, and as a result, we have to remember to treat our forecasts as what they really are: educated guesses that need constant refinement.

Automating the Forecasting

Our use of Excel in the previous examples was pretty straightforward. But you can automate that process by using Excel macros. And since you'll most likely be doing the same process repeatedly as your metric collection system churns out new usage data, you can benefit greatly by introducing some automation into this curve-fitting business. Other benefits can include the ability to integrate these forecasts into a dashboard, plug them into other spreadsheets, or put them into a database.

An open source program called *fityk* (http://fityk.sourceforge.net) does a great job of curve-fitting equations to arbitrary data, and can handle the same range of equation types as Excel. For our purposes, the full curve-fitting abilities of fityk are a distinct overkill. It was created for analyzing scientific data that can represent wildly dynamic datasets, not just growing and decaying data. While fityk is primarily a GUI-based application (see Figure 4-12), a command-line version is also available, called *cfityk*. This version accepts commands that mimic what would have been done with the GUI, so it can be used to automate the curve fitting and forecasting.



No	#F+#	Name
2	4+0	547.90
3	4+0	545.10
4	4+0	542.05
5	4+0	539.17
6	4+0	537.71
7	4+0	537.28

@6
512 points, 512 active.
Filename:

%_26 : Voigt

height	98.9568
center	526.787
gwidth	0.30185
shape	0.763907

```
#6: WSSR=52.9648 lambda=0.001 d(WSSR)=-0.000846312 (0.00159823%)
Fit converged.
Better fit found (WSSR = 52.9648, was 77.6317, -31.7743%).
=-> # the datasets from Markus Weigand
```

Figure 4-12. The fityk curve-fitting GUI tool

The command file used by cfityk is nothing more than a script of actions you can write using the GUI version. Once you have the procedure choreographed in the GUI, you'll be able to replay the sequence with different data via the command-line tool.

If you have a carriage return–delimited file of x-y data, you can feed it into a command script that can be processed by cfityk. The syntax of the command file is relatively straightforward, particularly for our simple case. Let's go back to our storage consumption data for an example.

In the code example that follows, we have disk consumption data for a 15-day period, presented in increments of one data point per day. This data is in a file called *storage-consumption.xy*, and appears as displayed here:

```
      1 14321.83119

      2 14452.60193

      3 14586.54003

      4 14700.89417

      5 14845.72223

      6 15063.99681

      7 15250.21164

      8 15403.82607

      9 15558.81815

     10 15702.35007

     11 15835.76298

     12 15986.55395

     13 16189.27423

     14 16367.88211

     15 16519.57105
```

The cfityk command file containing our sequence of actions to run a fit (generated using the GUI) is called *fit-storage.fit*, and appears as shown below:

```
# Fityk script. Fityk version: 0.8.2

 @0 < '/home/jallspaw/storage-consumption.xy'

 guess Quadratic

 fit

info formula in @0

quit
```

This script imports our x-y data file, sets the equation type to a second-order polynomial (quadratic equation), fits the data, and then returns back information about the fit, such as the formula used. Running the script gives us these results:

```
jallspaw:~]$cfityk ./fit-storage.fit
1> # Fityk script. Fityk version: 0.8.2
2>  @0 < '/home/jallspaw/storage-consumption.xy'
15 points. No explicit std. dev. Set as sqrt(y)
3>  guess Quadratic
New function %_1 was created.
4>  fit
Initial values:  lambda=0.001  WSSR=464.564
#1:  WSSR=0.90162  lambda=0.0001  d(WSSR)=-463.663  (99.8059%)
#2:      WSSR=0.736787      lambda=1e-05      d(WSSR)=-0.164833
(18.2818%)
#3:     WSSR=0.736763     lambda=1e-06     d(WSSR)=-2.45151e-05
(0.00332729%)
#4:     WSSR=0.736763     lambda=1e-07     d(WSSR)=-3.84524e-11
(5.21909e-09%)
Fit converged.
Better fit found (WSSR = 0.736763, was 464.564, -99.8414%).
5> info formula in @0
# storage-consumption
14147.4+146.657*x+0.786854*x^2
6> quit

bye...
```

We now have our formula to fit the data:

$0.786854x^2 + 146.657x + 14147.4$

Note how the result looks almost exactly as Excel's for the same type of curve. Treating the values for *x* as days and those for *y* as our increasing disk space, we can plug in our 25-day forecast, which yields the same results as the Excel exercise. Table 4-2 lists the results generated by cfityk.

Table 4-2. Same forecast as Table 4-1, curve-fit by cfityk

	Date	Disk Available (GB)	$y=0.786854x^2 + 146.657x + 14147.4$
33	08/27/05	20480.00	19843.97
34	08/28/05	20480.00	20043.34
35	08/29/05	20480.00	20244.29
36	08/30/05	20480.00	20446.81
37	08/31/05	20480.00	20650.91
38	09/01/05	20480.00	20856.58
39	09/02/05	20480.00	21063.83

Being able to perform curve-fitting with a cfityk script allows you to carry out forecasting on a daily or weekly basis within a cron job, and can be an essential building block for a capacity planning dashboard.

Safety Factors

Web capacity planning can borrow a few useful strategies from the older and better-researched work of mechanical, manufacturing, and structural engineering. These disciplines also need to base design and management considerations around resources and immutable limits. The design and construction of buildings, bridges, and automobiles obviously requires some intimate knowledge of the strength and durability of materials, the loads each component is expected to bear, and what their ultimate failure points are. Does this sound familiar? It should, because capacity planning for web operations shares many of those same considerations and concepts.

Under load, materials such as steel and concrete undergo physical stresses. Some have elastic properties that allow them to recover under light amounts of load, but fail under higher strains. The same concerns exist in your servers, network, or storage. When their resources reach certain critical levels—100 percent CPU or disk usage, for example—they fail. To pre-empt this failure, engineers apply what is known as a *factor of safety* to their design. Defined briefly, a factor of safety indicates some margin of resource allocated beyond the theoretical capacity of that resource, to allow for uncertainty in the usage.

While safety factors in the case of mechanical or structural engineering are usually part of the design phase, in web operations they should be considered as an amount of available resources that you leave aside, with respect to the ceilings you've established for each class of resource. This will enable those resources to absorb some amount of unexpected increased usage. Resources with which you should calculate safety factors include all the those discussed in Chapter 3: CPU, disk, memory, network bandwidth, even entire hosts (if you run a very large site).

For example, in Chapter 3 we stipulated 85 percent CPU usage as our upper limit for web servers, in order to reserve "enough headroom to handle occasional spikes." In this case, we're allowing a 15 percent margin of "safety." When making forecasts, we need to take these safety factors into account and adjust the ceiling values appropriately.

Why a 15 percent margin? Why not 10 or 20 percent? Your safety factor is going to be somewhat of a slippery number or educated guess. Some resources, such as caching systems, can also tolerate spikes better than others, so you may want to be less conservative with a margin of safety. You should base your safety margins on "spikes" of usage that you've seen in the past. See Figure 4-13.

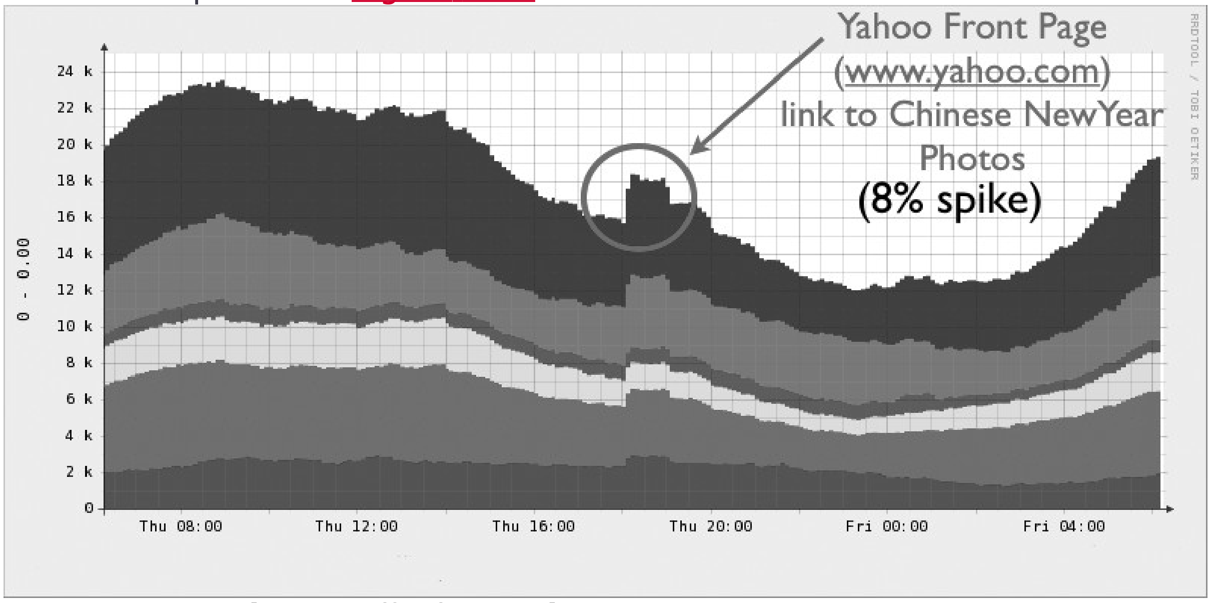


Figure 4-13. Spike in traffic from Yahoo Front Page

Figure 4-13 displays the effect of a typically-sized traffic spike Flickr experiences on a regular basis. It's by no means the largest. Spikes such as this one almost always occur when the front page of http://www.yahoo.com posts a prominent link to a group, a photo, or a tag search page on Flickr. This particular spike was fleeting; it lasted only about two hours while the link was up. It caused an eight percent bump in traffic to our photo servers. Seeing a 5–15 percent increase in traffic like this is quite common, and confirms that our 15 percent margin of safety is adequate.

Procurement

As we've demonstrated, with our resource ceilings pinpointed, we can predict when we'll need more of a particular resource. When we complete the task of predicting when we'll need more, we can use that timeline to gauge when to trigger the procurement process.

Your procurement pipeline is the process by which you obtain new capacity. It's usually the time it takes to justify, order, purchase, install, test, and deploy any new capacity. Figure 4-14 illustrates the procurement pipeline.

The tasks outlined in Figure 4-14 vary from one organization to another. In some large organizations, it can take a long time to gain approvals to buy hardware, but delivery can happen quickly. In a startup, approvals may come quickly, but the installation likely proceeds more slowly. Each situation will be different, but the challenge will remain the same: estimate how long the entire process will take, and add some amount of comfortable buffer to account for unforeseen problems. Once you have an idea of what that buffer timeline is, you can then work backward to plan capacity.

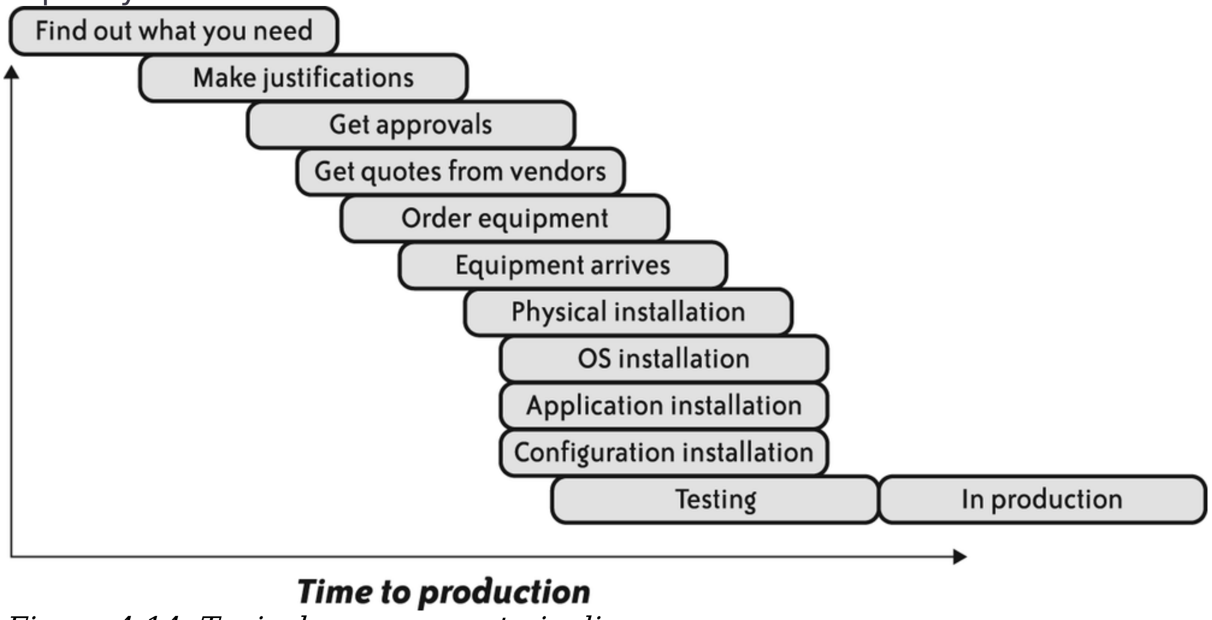


Figure 4-14. Typical procurement pipeline

In our disk storage consumption example, we have current data on our disk consumption up to 8/15/05, and we estimate we'll run out of space on

8/30/05. You now know you have exactly two weeks to justify, order, receive, install, and deploy new storage. If you don’t, you’ll run out of space and be forced to trim that consumption in some way. Ideally, this two-week deadline will be long enough for you to bring new capacity online.

Procurement Time: The Killer Metric

Obviously, the *when* of ordering equipment is just as important as the *what* and *how much*. Procurement timelines outlined above hint at how critical it is to keep your eye on how long it will take to get what you need into production. Sometimes external influences, such as vendor delivery times and physical installation at the data center can ruin what started out to be a perfectly timed integration of new capacity.
Startups routinely order servers purely out of the fear they’ll be needed. Most newly launched companies have developers to work on the product and don’t need to waste money on operations-focused engineers. The developers writing the code are most likely the same people setting up network switches, managing user accounts, installing software, and wearing whatever other hats are necessary to get their company rolling. The last thing they want to worry about is running out of servers when they launch their new, awesome website. Ordering more servers as needed can be rightly justified in these cases, because the hardware costs are more than offset by the costs of preparing a more streamlined and detailed capacity plan.

But as companies mature, optimizations begin to creep in. Code becomes more refined. The product becomes more defined. Marketing starts to realize who their users are. The same holds true for the capacity management process; it becomes more polished and accurate over time.

Just-In-Time Inventory

Toyota Motors developed the first implementations of a just-in-time inventory practice. It knew there were large costs involved to organize, store, and track excess inventory of automobile parts, so it decided to reduce that “holding” inventory and determine exactly when it needed parts. Having inventory meant wasting money. Instead of maintaining a massive warehouse filled with the thousands of parts to make its cars, Toyota would only order and stock those parts as they were needed. This reduced costs tremendously and gave Toyota a competitive advantage in the 1950s. Just-in-time inventory practice is now part of any modern manufacturing effort.

The costs associated with having auto parts lying around in a warehouse can be seen analogous to having servers installed before you *really* need them. Rack space and power consumption in a data center cost money, as

does the time spent installing and deploying code on the servers. More important, you risk suffering economically as a result of the aforementioned Moore’s Law, which if your forecasts allow it, should motivate you to buy equipment later, rather than sooner.
Once you know when your current capacity will top out, and how much capacity you’ll need to get through to the next cycle of procurement, you should take a few lessons from the just-in-time inventory playbook, whose sole purpose it is to eliminate waste of time and money in the process.

Here are some of the steps in our typical procurement process you’ll want to pay attention to, and streamline:

1. Determine your needs
   You know how much load your current capacity can handle, because you’ve followed the advice in Chapter 3 to find their ceilings and are measuring their usage constantly. Take these numbers to the curve-fitting table and start making crystal ball predictions. This is fundamental to the capacity planning process.
2. Justify purchases
   Add some color and use attention-grabbing fonts on the graphs you just made in the previous step, because you’re going to show them to the people who will approve the hardware purchases you’re about to make. Spend as much time as you need to ensure your money-handling audience understands why you’re asking for capacity, why you’re asking for it now, and why you’ll be coming back later asking for more. Be very clear in your presentations about the downsides of insufficient capacity.
3. Solicit quotes from vendors
   Vendors want to sell you servers and storage; you want to buy servers and storage—all is balanced in the universe. Why would you choose vendor A over vendor B? Because vendor A might help alleviate some of the fear normally associated with ordering servers, through such practices as quick turnarounds on quotes and replacements, discounts on servers ordered later, or discounts tied to delivery times.
4. Order equipment
   Can you track your order online? Do you have the phone number (gasp!) of a reliable human who can tell you where your equipment is at all times? Does the data center know the machines are coming, and have they factored that into their schedule?
5. Physical installation
   How long will it take for the machines to make the journey from a loading dock into a rack, and cabled up to a working switch? Does the data center staff need to get involved, or are you racking machines yourself? Are there enough rack screws? Power drill batteries? Crossover cables? How long is this entire process going to take?
6. OS/application/configuration installation

In the next chapter, we'll talk about deployment scenarios that involve automatic OS installation, software deployment, and configuration management. However, just because it's automated doesn't mean it doesn't take time and that you shouldn't be aware of any problems that can arise.

7. Testing
   Do you have a QA team? Do you have a QA environment? Testing your application means having some process by which you can functionally test all the bits you need to make sure everything is in its right place. Entire books are written on this topic; I'll just remind you that it's a necessary step in the journey toward production life as a server.
8. Deploy your new equipment
   It's not over until the fat server sings. Putting a machine into production should be straightforward. When doing so, you should use the same process to measure the capacity of your new servers as outlined in the Chapter 3. Maybe you'll want to ramp up the production traffic the machine receives by increasing its weight in the load-balanced pool. If you know this new capacity relieves a bottleneck, you'll want to watch any effect that has on your traffic.

The Effects of Increasing Capacity

All of the segments within your infrastructure interact in various ways. Clients make requests to the web servers, which in turn make requests to databases, caching servers, storage, and all sorts of miscellaneous components. Layers of infrastructure work together to respond to users by providing web pages, pieces of web pages, or confirmations that they've performed some action, such as uploading a photo.

When one or more of those layers encounters a bottleneck, you bring your attention to bear, figure out how much more capacity you need, and then deploy it. Depending on how bottlenecked that layer or cluster is, you may find you'll see second-order effects of that new deployment, and end up simply moving the traffic jam to yet another part of your architecture.

For example, let's assume your website involves a web server and a database. One of the ways organizations can help scale their application is to cache computationally expensive database results. Deploying something like *memcached* can allow you to do this. In a nutshell, it means for certain database queries you choose, you can consult an in-memory cache before hitting the database. This is done primarily for the dual purpose of speeding up the query and reducing load on the database server for results that are frequently returned.
The most noticeable benefit is queries that used to take seconds to process might take as little as a few milliseconds, which means your web server will be able to send the response to the client more quickly. Ironically, there's a side effect to this; when users are not waiting for

pages as long, they have a tendency to click on links faster, causing more load on the web servers. It’s not uncommon to see *memcached* deployments turn into web server capacity issues rather quickly.

Long-Term Trends

Now you know how to apply the statistics collected in Chapter 3 to immediate needs. But you may also want to view your site from a more global perspective—both in the literal sense (as your site becomes popular internationally), and in a figurative sense, as you look at the issues surrounding the product and the site’s strategy.

Traffic Pattern Changes

As mentioned earlier, getting to know the peaks and valleys of your various resources and application usage is paramount to predicting the future. As you gain more and more history with your metrics, you may be able to perceive more subtle trends that will inform your long-term decisions.

For example, let’s take a look at Figure 4-15, which illustrates a typical traffic pattern for a web server.

Figure 4-15 shows a pretty typical U.S. daily traffic pattern. The load rises slowly in the morning, East Coast time, as users begin browsing. These users go to lunch as West Coast users come online, keeping up the load, which finally drops off as people leave work. At this point, the load drops to only those users browsing over night.

As your usage grows, you can expect this graph to grow vertically as more users visit your site during the same peaks and valleys. But if your audience grows more internationally, the bump you see every day will widen as the number of active user time zones increases. As seen in Figure 4-16, you may even see distinct bumps after the U.S. drop-off if your site’s popularity grows in a region further away than Europe.

Figure 4-16 displays two daily traffic patterns, taken one year apart, and superimposed one on top of the other. What once was a smooth bump and decline has become a two-peak bump, due to the global effect of popularity.

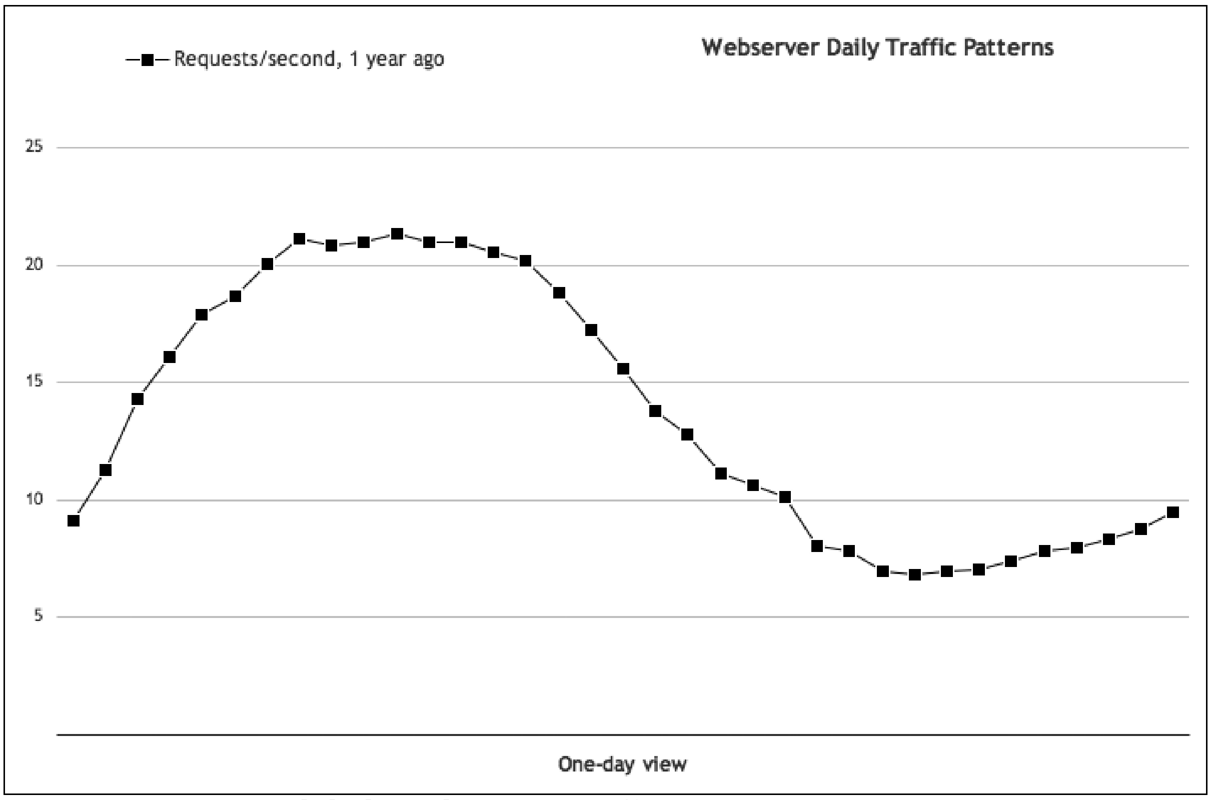


Figure 4-15. Typical daily web server traffic pattern

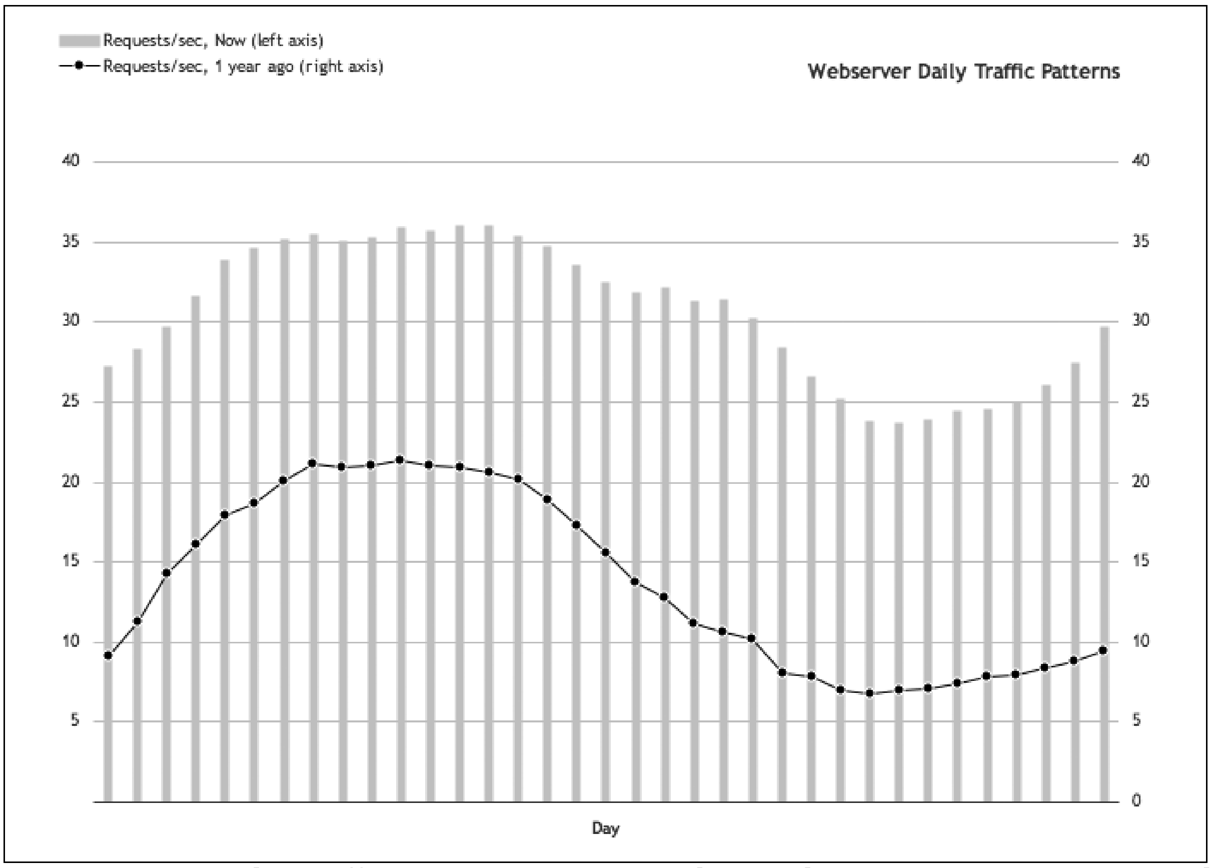


Figure 4-16. Daily traffic patterns grow wider with increasing international usage

Of course, your product and marketing people are probably very aware of the demographics and geographic distribution of your audience, but tying this data to your system's resources can help you predict your capacity needs.

Figure 4-16 also shows that your web servers must sustain their peak traffic for longer periods of time. This will indicate when you should schedule any maintenance windows to minimize the effect of downtime or degraded service to the users. Notice the ratio between your peak and your low period has changed as well. This will affect how many servers you can stand to lose to failure during those periods, which is effectively the ceiling of your cluster.
It's important to watch the change in your application's traffic pattern, not only for operational issues, but to drive capacity decisions, such as whether to deploy any capacity into international data centers.

ORA: FORECASTING CONSIDERATIONS FOR MULTIPLE DATA CENTERS
When your site's growth dictates serving content out of multiple data centers, you might notice certain geographic traffic patterns can emerge you may not have seen with a single data center. At the time of this writing, Flickr serves photos from eight different locations in the U.S., and will soon be serving photos from European and Asian locations as well. Our data centers in the U.S. are split between the east and west coasts, and users pull content from the data center to which they are geographically closest. Photos are distributed into what we term *photo farms*. A farm is made up of a mirrored pair of data centers, one on each coast. We currently have four farms (thus eight different data centers), with each farm containing a unique set of photos. We store each photo in the two locations for redundancy, in the event of emergency, or if one of a farm's data centers needs to be taken offline for maintenance.
When we first launched our second location, we noticed the east coast data center received as much as 65–70 percent more traffic at peak than its west coast counterpart. This was easily explained, as at the time our European users were a much more engaged audience than our Asian users, and since the U.S. east coast is closer to Europe, the usage was commensurately higher. In addition, we noticed the west coast data centers received considerably more requests for the larger, original sizes of photos than the east coast. We've attributed this to home broadband connections in Asia having higher bandwidth limits, so they're accustomed to downloading larger amounts of data. The usage gap has narrowed as the Asian user base has become more engaged, but overall activity is still higher in the east.

What this meant was our peaks and valleys differed for each side of each farm, and therefore our forecasts needed to be adjusted when we planned for growth. As just mentioned, the architecture for each data center in a farm dictates it must be able to handle 100 percent of the traffic for entire farm in the event its partner data center falls out of operation. Therefore, capacity forecasts need to be based on the cumulative peaks of both data centers in a farm.

As I alluded to in Chapter 3 (“Knowing Your Waves”), when you deploy capacity to multiple data centers, usage patterns can become more complex. You’ll need to take that into consideration when forecasting capacity.

Application Usage Changes and Product Planning

A good capacity plan not only relies on system statistics such as peaks and valleys, but user behavior as well. How your users interact with your site is yet another valuable vein of data you should mine for information to help keep your crystal ball as clear as possible.

If you run an online community, you might have discussion boards in which users create new topics, make comments, and upload media such as video and photos. In addition to the previously discussed system-related metrics, such as storage consumption, video and photo processing CPU usage, and processing time, some other metrics you might want to track are:

- Discussion posts per minute
- Posts per day, per user
- Video uploads per day, per user
- Photo uploads per day, per user

Application usage is just another way of saying *user engagement*, to borrow a term from the product and marketing folks.

Recall back to our database-planning example. In that example, we found our database ceiling by measuring our hardware’s resources (CPU, disk I/O, memory, and so on), relating them to the database’s resources (queries per second, replication lag) and tying those ceilings to something we can measure from the user interaction perspective (how many photos per user are on each database).

This is where capacity planning and product management tie together. Using your system and application statistics histories, you can now predict with some (hopefully increasing) degree of accuracy what you’ll need to meet future demand. But your history is only part of the picture. If your product team is planning new features, you can bet they’ll affect your capacity plan in some way.

Historically, corporate culture has isolated product development from engineering. Product people develop ideas and plans for the product, while engineering develops and maintains the product once it’s on the market. Both groups make forecasts for different ends, but the data used in those forecasts should tie together.

One of the best practices for a capacity planner is to develop an ongoing conversation with product management. Understanding the timeline for new features is critical to guaranteeing capacity needs don’t interfere with product improvements. Having enough capacity is an engineering requirement, in the same way development time and resources are.

Iteration and Calibration

Producing forecasts by curve-fitting your system and application data isn’t the end of your capacity planning. In order to make it accurate, you need to revisit your plan, re-fit the data, and adjust accordingly.

Ideally, you should have periodic reviews of your forecasts. You should check how your capacity is doing against your predictions on a weekly, or even daily, basis. If you know you’re nearing capacity on one of your resources and are awaiting delivery of new hardware, you might keep a much closer eye on it. The important thing to remember is your plan is going to be accurate only if you consistently re-examine your trends and question your past predictions.

As an example, we can revisit our simple storage consumption data. We made a forecast based on data we gleaned for a 15-day period, from 7/26/05 to 8/09/05. We also discovered that on 8/30/2005 (roughly two weeks later), we expected to run out of space if we didn’t deploy more storage. More accurately, we were slated to reach 20,446.81 GB of space, which would have exceeded our total available space is 20,480 GB.

How accurate was that prediction? Figure 4-17 shows what actually happened.

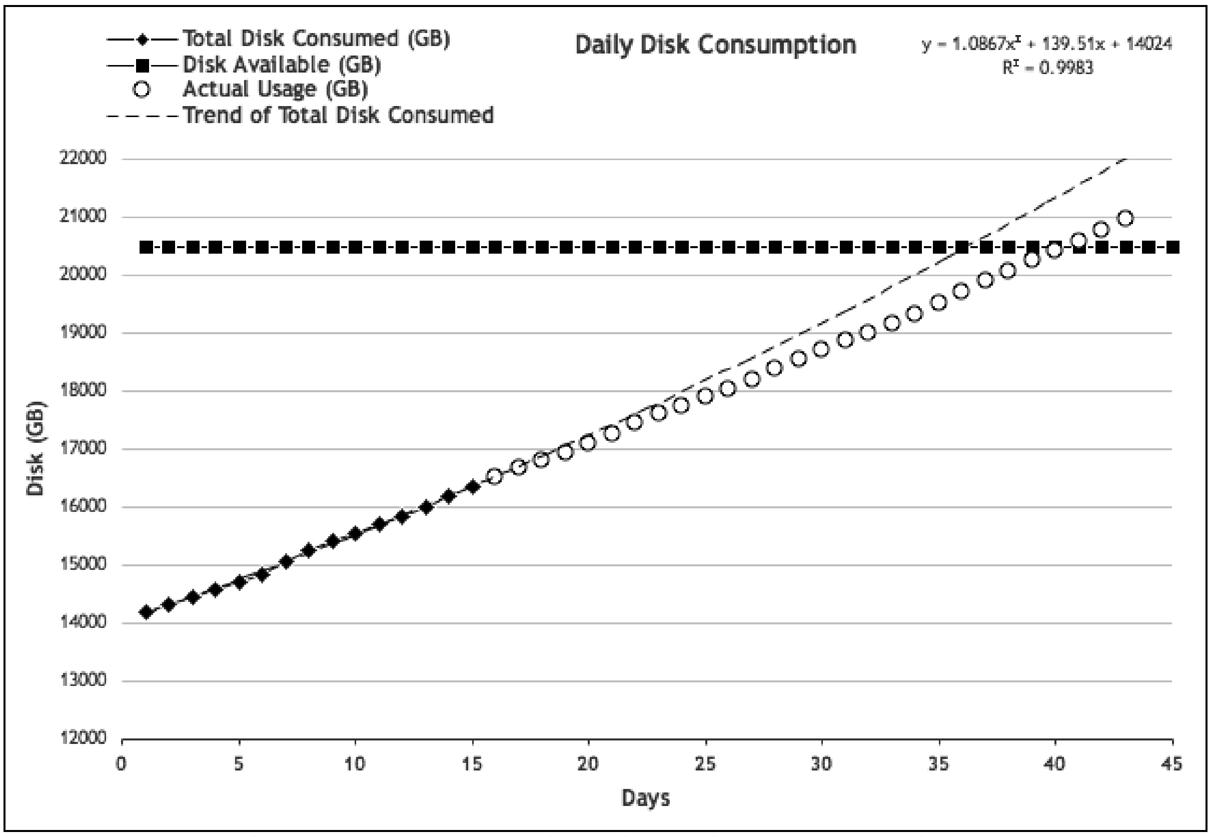


Figure 4-17. Disk consumption: predicted trend versus actual

As it turned out, we had a little more time than we thought—about four days more. We made a guess based on the trend at the time, which ended up being inaccurate but at least in favor of allowing more time to integrate new capacity. Sometimes, forecasts can either widen the window of time (as in this case) or tighten that window.

This is why the process of revisiting your forecasts is critical; it's the only way to adjust your capacity plan over time. Every time you update your capacity plan, you should go back and evaluate how your previous forecasts fared.

Since your curve-fitting and trending results tend to improve as you add more data points, you should have a moving window with which you make your forecasts. The width of that forecasting window will vary depending on how long your procurement process takes.

For example, if you know that it's going to take three months on average to order, install, and deploy capacity, then you'd want your forecast goal to be three months out, each time. As the months pass, you'll want to add the influence of most recent events to your past data and recalculate your predictions, as is illustrated in Figure 4-18.

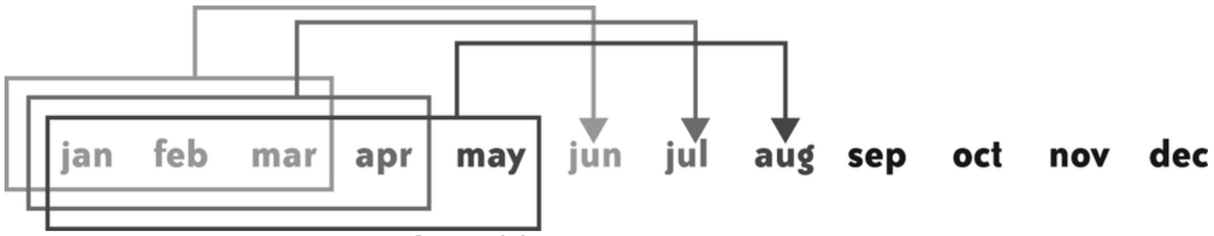


Figure 4-18. A moving window of forecasts

Best Guesses

This process of plotting, prediction, and iteration can provide a lot of confidence in how you manage your capacity. You'll have accumulated a lot of data about how your current infrastructure is performing, and how close each piece is to their respective ceilings, taking into account comfortable margins of safety. This confidence is important because the capacity planning process (as we've seen) is just as much about educated guessing and luck as it is about hard science and math. Hopefully, the iterations in your planning process will point out any flawed assumptions in the working data, but it should also be said the ceilings you're using could become flawed or obsolete over time as well.

Just as your ceilings can change depending on the hardware specifications of a server, so too can the actual metric you're assuming is your ceiling. For example, the defining metric of a database might be disk I/O, but after upgrading to a newer and faster disk subsystem, you might find the limiting factor isn't disk I/O anymore, but the single gigabit network card you're using. It bears mentioning that picking the right metric to follow can be difficult, as not all bottlenecks are obvious, and the metric you choose can change as the architecture and hardware limitations change.

During this process you might notice seasonal variations. College starts in the fall, so there might be increased usage as students browse your site for materials related to their studies (or just to avoid going to class). As another example, the holiday season in November and December almost always witness a bump in traffic, especially for sites involving retail sales. At Flickr, we see both of those seasonal effects.

Taking into account these seasonal or holiday variations should be yet another influence on how wide or narrow your forecasting window might be. Obviously, the more often you recalculate your forecast, the better prepared you'll be, and the sooner you'll notice variations you didn't expect.

Diagonal Scaling Opportunities

As I pointed out near the beginning of the chapter, predicting capacity requires two essential bits of information: your ceilings and your historical data. Your historical data is etched in stone. Your ceilings are not, since each ceiling you have is indicative of a particular hardware configuration. Performance tuning can hopefully change those ceilings for the better, but upgrading the hardware to newer and better technology is also an option.

As I mentioned at the beginning of the book, new technology (such as multicore processors) can dramatically change how much horsepower you can squeeze from a single server. The forecasting process shown in this chapter allows you to not only track where you're headed on a per-node basis, but also to think about which segments of the architecture you might possibly move to new hardware options.

As discussed, due to the random access patterns of Flickr's application, our database hardware is currently bound by the amount of disk I/O the servers can manage. Each database machine currently comprises six disks in a RAID10 configuration with 16 GB of RAM. A majority of that physical RAM is given to MySQL, and the rest is used for a filesystem cache to help with disk I/O. This is a hardware bottleneck that can be mitigated in a variety of ways, including at least:

- Spreading the load horizontally across many six-disk servers (current plan)
- Replacing each six-disk server with hardware containing more disk spindles
- Adding more physical RAM to assist both the filesystem and MySQL
- Using faster I/O options such as Solid-State Disks (SSDs)

Which of these options is most likely to help our capacity footprint? Unless we have to grow the number of nodes very quickly, we might not care right now. If we have accurate forecasts for how many servers we'll need in our current configuration, we're in a good place to evaluate the alternatives.

Another long-term option may be to take advantage of the bottlenecks we have on those machines. Since our disk-bound boxes are not using many CPU cycles, we could put those mostly idle CPUs to use for other tasks, making more efficient use of the hardware. We'll talk more about efficiency and virtualization in Appendix A.

Summary

Predicting capacity is an ongoing process that requires as much intuition as it does math to help you make accurate forecasts. Even simple web applications need to be attended, and some of this crystal ball work can be tedious. Automating as much of the process as you can will help you stay ahead of the procurement process. Taking the time to connect your metric collection systems to trending software, such as cfityk will prove to be invaluable as you develop a capacity plan that is easily adaptable. Ideally, you'll want some sort of a capacity dashboard that can be referred to at any point in time to inform purchasing, development, and operational decisions.

The overall process in making capacity forecasts is pretty simple:

1. Determine, measure, and graph your defining metric for each of your resources.
   Example: disk consumption
2. Apply the constraints you have for those resources.
   Example: total available disk space
3. Use trending analysis (curve fitting) to illustrate when your usage will exceed your constraint.
   Example: find the day you'll run out of disk space

Chapter 5. Deployment

ONCE YOU HAVE AN IDEA OF HOW MUCH CAPACITY YOU'LL NEED FOR FUTURE GROWTH AND HAVE PURCHASED the hardware, you'll need to physically install it and deploy it into production.
Historically, deployment has been viewed as a headache. Installing the operating system and application software, making sure all of the right settings are in place, and loading your website's data—all these tedious steps must be done in order to integrate new hardware that's fresh out of the crate. Fortunately, the pain of repeating these steps over and over has inspired an entire category of software: automated installation and configuration tools.[1]

Automated Deployment Philosophies

Although various automatic installation and configuration tools differ in their implementation and execution, most of them share the same design philosophy. Just as with monitoring and metric-collection tools, many of these concepts and designs originated in the high-performance computing (HPC) field. Because HPC and web operations have similarities in their infrastructure, the web operations community has adopted many of these tools and approaches.

Goal: Minimize Time to Provision New Capacity

The time needed to acquire, install, and provision new hardware must be factored into your calculations as you determine when you're going to run out of capacity. If your capacity will be exhausted in six weeks, and it takes you three weeks to add new hardware, you only have three weeks of breathing room. Automated deployment and configuration minimizes the time spent on the phase of the process over which you have the most control—integrating machines onto your network and beginning operations.

Goal: All Changes Happen in One Place

When making changes to hosts, it's preferable to have a central location from which to push changes appropriate to the servers you're affecting. Having a central location provides a "control tower" from which to manage all aspects of your infrastructure. Unlike server architectures, in which distributed resources help with horizontal scaling, centralized configuration and management environments yield several advantages:

- Version control can be used for all configurations: OS, application, or otherwise. RCS/CVS/Subversion and others are

used to track the “who, what, when, and why” of each change to the infrastructure.
- Replication and backup of installation and configuration files is easier to manage.
- An aggregated configuration and management logging system is an ideal troubleshooting resource.
- This centralized management environment makes an ideal place to keep hardware inventory, particularly if you want to have different configuration settings for different hardware.

This is not to suggest that your configuration, installation, monitoring, and management setup should be kept on a single server. Each of these deployment components demands specific resources. Growth over time would simply overwhelm a single machine, rendering it a potential single point of failure. Separate these components from the rest of your infrastructure. Monitoring and metric collection can reside on one server; configuration management and log aggregation on another. See Figure 5-1 for an example of a typical installation, configuration, and management architecture.

Goal: Never Log In to an Individual Server (for Management)

Restrict logging into an individual server to troubleshooting or forensic work only. All configuration or management should be done from the centralized management servers. This keeps your production changes in an auditable and controlled environment.

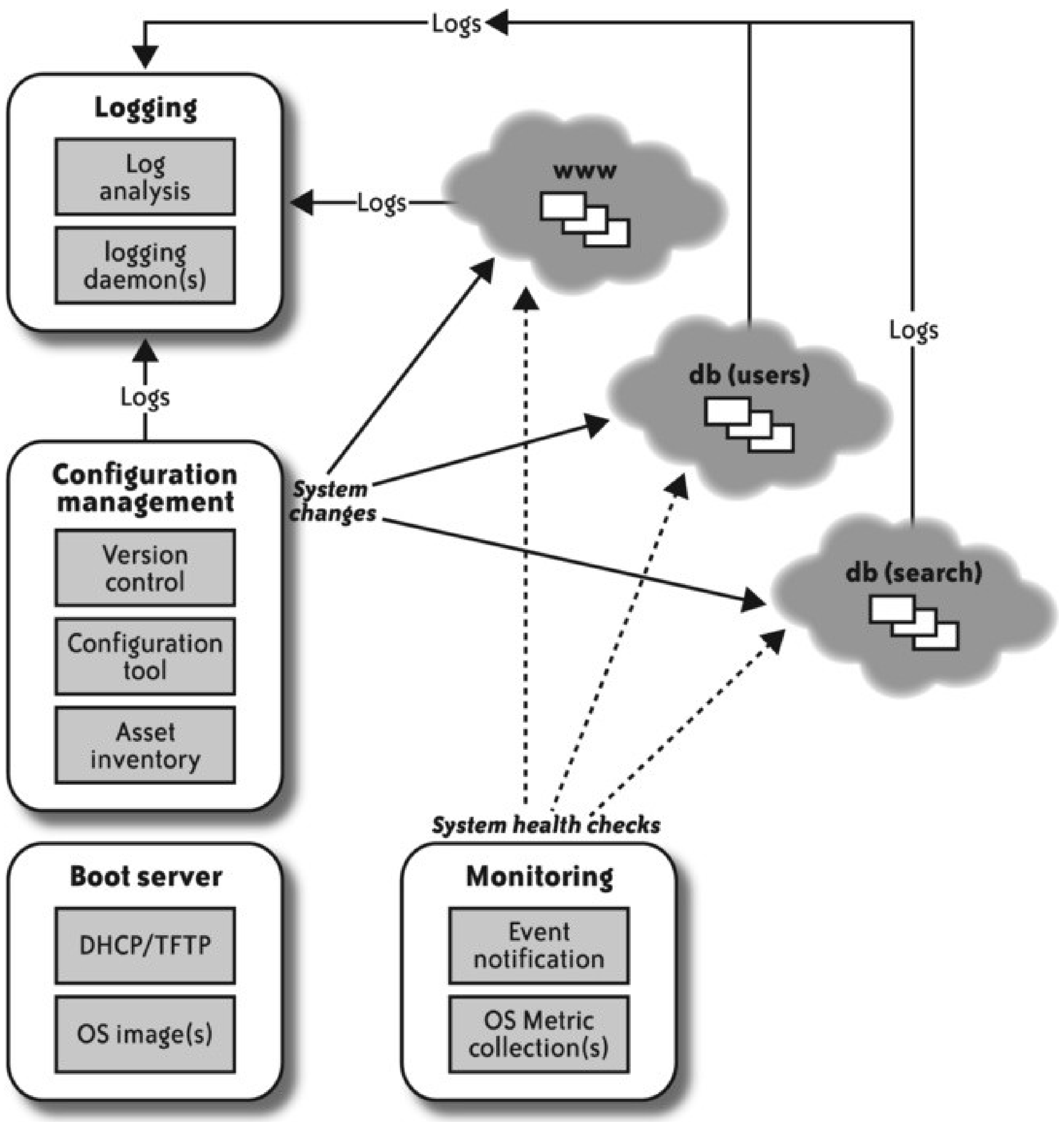


Figure 5-1. Typical cluster management scenario

Goal: Have New Servers Start Working Automatically

This is the holy grail of all deployment systems. You should be able to order hardware, add it to your inventory-management system, turn it on, have it automatically install all required software and register itself with your monitoring and trending systems, and then get down to work, all without administrator intervention.

Maintain Consistency for Easier Troubleshooting

Coordination of troubleshooting efforts during outages or emergencies is critical. Configuration management tools effectively enable that coordination. Consistent OS and software configuration across a cluster of identically deployed servers eliminates configuration inconsistencies as a source of problems. Troubleshooting in large production environments is difficult enough without needing to isolate configuration discrepancies between a handful of servers.

When tracking down a bug, the ability to quickly view all the changes that occurred between the last known good state and the present (bad) state can be invaluable. When more than one person is working on an issue, a centralized and consistent overview of all of the changes is crucial. With version control, changes are immediately apparent and tasks aren't repeated. Nor are they assumed to have been completed or otherwise mixed up with other tasks.

ORA: HOMOGENIZE HARDWARE TO HALT HEADACHES

In addition to making your software configurations consistent between servers that perform the same role, it's valuable to have consistency at the hardware level as well. At Flickr, we have two basic server types: a multiple-disk machine for I/O intensive tasks, and a single-disk machine for CPU-intensive jobs (for which most of the working set fits in memory). Limiting the number of hardware types has a number of advantages:

- It reduces variability between machines, which simplifies troubleshooting.
- It minimizes the number of different spare parts you need to keep on hand and facilitates cannibalizing dead machines for parts.
- It simplifies automated deployment and configuration because you don't need to create numerous special-case configurations for different hardware.

Virtualized infrastructures can take this a step further. You can buy racks of identical servers and allocate memory, CPU, and high-performance remote storage to virtual machines based on application need.

Automated Installation Tools

Before you can even begin to worry about configuration management, you need to get your servers to a state in which they can be configured. You want a system that can automatically (and repetitively) install your OS of choice. Many such systems have been developed over the years, all employing similar techniques.

There are two basic approaches to the task of imaging new machines. Most OS vendors offer a package-based installer option, which performs the normal installation process in a non-interactive fashion. It provides the installer with a configuration file that specifies the packages to be installed. Examples include Solaris Jumpstart, Red Hat Kickstart, and Debian FAI.

Many third-party products take a disk-image approach. A *gold client* image is prepared on one machine and replicated byte-for-byte onto

newly imaged hosts. Often, a single image is used for every server in the infrastructure, with hosts only differing in the services that are configured and running. SystemImager is a product that uses this approach.
Each method has advantages. Package-based systems provide accountability; every file installed is guaranteed to belong to a package, and package management tools make it easy to quickly see what's installed. You can get the same result with disk image systems by installing only packaged files. The temptation to muck about with the gold client filesystem directly can lead to confusion down the road.

On the other hand, image-based systems tend to be faster to install. The installer merely has to create a filesystem and dump the image onto it, rather than download many packages, calculate dependencies, and install them one by one. Some products, such as SystemImager, even support parallel installs to multiple clients by streaming the disk images via multicast or BitTorrent.

Preparing the OS Image

Most organizations aren't happy with the operating system their vendor installs. Default OS installs are notoriously inappropriate for production environments because they are designed to run on as many different hardware platforms as possible. They usually contain packages you don't need and typically are missing those that you do. As a result, most companies create custom OS images suitable for their specific needs.

For a package-based system, the OS image is specified by preparing a configuration file that lists the set of packages to be installed. You can simply dump the package list from an existing server and tweak it as desired. Then you put the configuration file in the appropriate place on the boot server, and you're ready to go.

For an image-based system, it's slightly more involved. Normally, a server is set aside to be the gold client that provides the template for the deployed image. This can be a physical server set aside for the purpose, or even a virtual machine. You perform a normal OS installation and install all the software required in your infrastructure. The client image is then copied to the deployed server.

The Installation Process

If you're installing more than a handful of machines, installations with physical boot media such as CD-ROMs quickly become tedious and require someone to be physically present at the data center. You're going to want to set up a network boot server, which in the case of PC hardware usually means PXE (Figure 5-2), or Pre-boot Execution Environment.

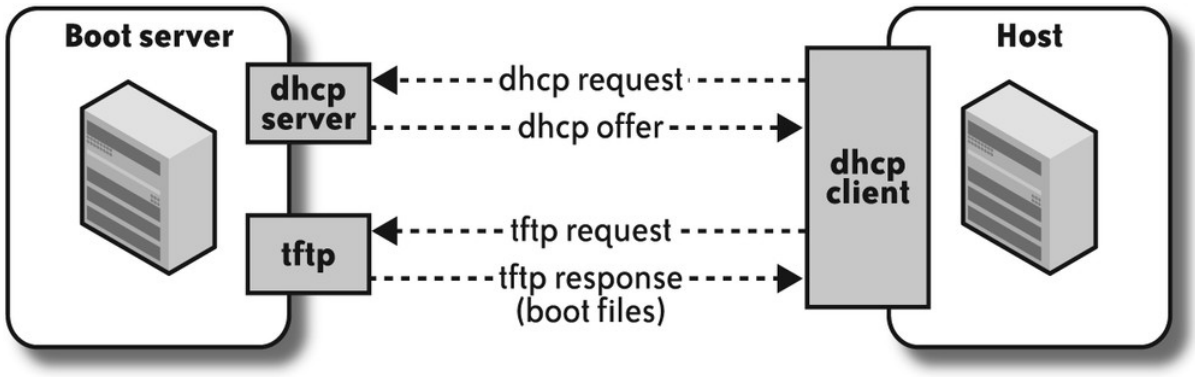


Figure 5-2. Basic steps of the PXE booting process

PXE-based installs are performed by a number of services working together. PXE firmware on the client requests its environment via DHCP. A DHCP server provides the information required to fetch the boot image (IP address, boot server, and image name). Finally, the client fetches the boot image from the boot server via TFTP.

For automated installation systems, the boot image consists of a kernel, plus a ramdisk containing the installer program. (This differs from diskless client setups, where the boot image contains the production kernel the client uses to mount a network volume as its root filesystem.)

The installer then determines what type of host is being installed and formats the local filesystems appropriately. There are several methods for mapping hardware profiles to hosts, typically they involve assigning configurations based on hostname or MAC address. For example, Kickstart passes the configuration file name as part of the DHCP options and fetches it from the TFTP server, while SystemImager has a configuration file stored on the image server that maps image types to hostnames.

The installer then installs the OS from the network onto the newly formatted volumes. As it pertains to a package-based installer, this means copying package files from a network repository (for example, apt or yum). For image-based systems, the OS image is dumped directly onto the local volumes, usually via rsync or a similar program.

Once the OS image is installed, the PXE server marks the host as installed. Typically, this is done by replacing the host’s bootfile with a stub that instructs the host to boot from the local disk. The machine is restarted, and boots normally.

As shipped, most automated deployment systems require some manual configuration. DHCP server configurations need to be created, and hostnames mapped to roles. However, your inventory management system should have all the information about a machine required to boot it: MAC address, role, IP address, and hostname. With a bit of clever scripting, you can automatically generate the required configuration from your asset database.

Once this is set up, provisioning new servers is as simple as entering their details into inventory management, racking them, and powering them up. Reformatting a machine is as simple as assigning it a new role in the database and rebooting it. (Normally, a reinstall is required only if the disk layout changes. Otherwise, the server can simply be reconfigured.)

The deployment beast has been tamed.

Automated Configuration

Now that your machines are up on the network, it’s time to configure them to do their jobs. Configuration management systems help with this task in the following ways:

- They let you organize your configuration files into useful subsystems, which you can combine in various ways to build production systems.

  ORA: INVENTORY MANAGEMENT

  Once your site grows beyond a handful of boxes, you will need a system to keep track of your hardware. Inventory management systems can be as simple as a spreadsheet or as elaborate as a full-blown web application backed by a database. These systems have a frustrating tendency to drift out of sync with reality as hardware is moved, decommissioned, or repurposed.

  A way out of this trap is to make your inventory management system a part of your deployment infrastructure. By using information from your inventory management to direct your installs and configure deployments, you change it from a snapshot of your infrastructure as you *think* it is, to a source of truth that guides and instructs your systems.

  One system to take this approach is iClassify. It supports automatic registration, as well as automated and manual classification. iClassify also integrates with Puppet for configuration management, and Capistrano for ad-hoc management of servers.

- They put all the information about your running systems in one place, from which it can easily be backed up or replicated to another site.
- They extract institutional knowledge out of your administrator’s head and place it into a form that can be documented and reused.

A typical configuration management system consists of a server in which configurations are stored, and a client process, which runs on each host and requests a configuration from the server. In an infrastructure with automated deployment, the client is run as part of the initial install or immediately after the initial boot into the new OS.

After the initial configuration, a scheduled task on the client host periodically polls the server to see if any new configuration is available. Automating these checks ensures every machine in your infrastructure is always running the latest configuration.

Defining Roles and Services

You have a shiny new configuration management system installed. How do you actually use it? The best way to attack the problem is to divide and conquer. *Services* (collections of related software and configurations) are the atoms, and *roles* (the types of machines in your infrastructure) are the molecules. Once you have a robust set of services defined, it will be easy to shuffle existing services into alternative combinations to serve new roles or to split an existing role into more specialized configurations.
First, go through all of the machines in your infrastructure (or planned infrastructure) and identify the roles present. A role is a particular type of machine that performs a particular task. For a website, your list will include roles like “web server” and “database.”

Next, go through each role and determine which services need to be present on each instance of the role for the instance to be able to do its job. A service in this sense is not just an OS-level program like “httpd.” For example, the http server service would include not only the httpd package and its configuration, but also settings for any metrics, health checks, or associated software that runs on a machine serving web pages.

As you go through your roles, try to identify services that may be common to multiple roles. For example, every server is likely to require a remote login service such as sshd. By identifying these common services, you can create a single set of configuration routines that can be used over and over in roles you’re deploying now, and in new ones that will emerge as your site grows.

An Example: Splitting Off Static Web Content

Suppose you have a cluster of four web servers. Each machine serves a combination of static content and dynamic pages generated by PHP. You have Apache’s MaxClients setting tuned down to 80 simultaneous connections to ensure the machines don’t overrun available memory and initiate swapping. The web server role might look similar to Figure 5-3.

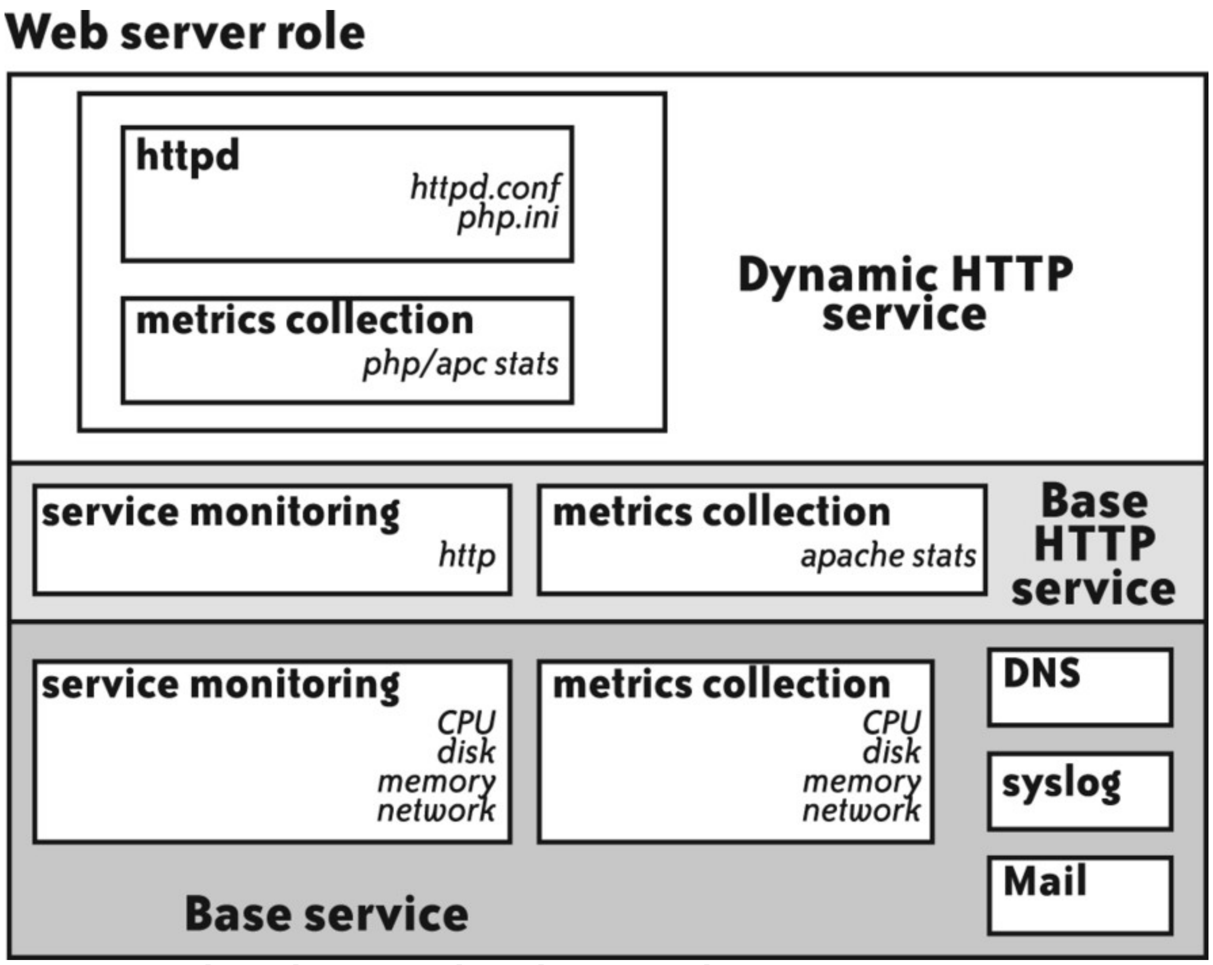


Figure 5-3. The web server role and associated services

You realize in a flash of insight that you could serve more simultaneous clients by splitting your cluster into two roles: one with the current configuration for dynamic content, and one with a stripped-down Apache with a larger MaxClients to serve only static content.

First, you split out services common to both roles into a separate service called base_http. This service includes settings that any web server should have, such as an HTTP health check and metrics pulled from Apache's status module.

Next, you create two new services. The dynamic http service contains your original Apache and PHP configurations. The static http service is configured with a simplified *httpd.conf* with a larger client cap and no PHP module.
You then define roles for each server type by combining these services. The new roles are depicted in Figure 5-4.

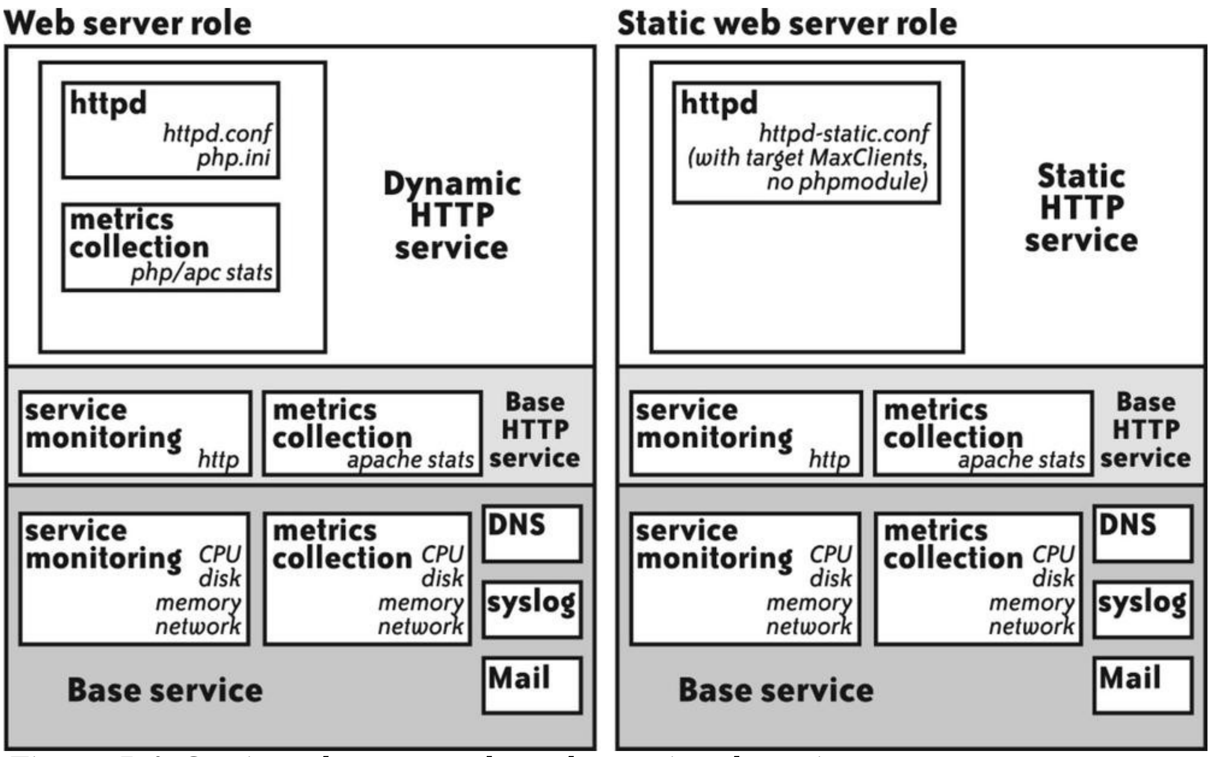


Figure 5-4. Static web server role and associated services

Now that the new role is defined, you can either go into your inventory management system and assign one or more existing web server machines to the static_webserver role, or deploy new hardware with the new role. If you decide to add more metrics or healthchecks, which are applicable to both roles in the future, you can put them in the base_http service, and both roles will inherit them.

User Management and Access Control

User management and access control require special consideration. How do you control which users have access to which portions of your system?

In fact, there are several ways. Network-based authentication and authorization services such as LDAP are popular. Users and groups can be granted access to individual machines or host groups in one place. Permission and password changes are propagated automatically. On the downside, these systems represent yet another service that needs to be provisioned, monitored, and scaled as your infrastructure grows.

Alternately, it’s possible to use your configuration management system to install user accounts on a host or role basis by defining services that make the appropriate changes to the system authentication databases. This is straightforward if you already have configuration management in place.

However, with such a system, password changes and access additions and revocations may not be applied to all servers simultaneously.

Additionally, if automated configuration updates are broken on a host, that host may not receive the latest access configuration at all, which is an obvious security concern.

Both of these setups can be made to work. Which one is most appropriate for your infrastructure depends on your existing authentication systems, the number of users involved, and the frequency with which changes are made. If you are already using LDAP elsewhere in your organization, that may be the natural choice. If you have a small number of users and changes are infrequent, a package-based system may be appropriate.

Ad Hockery

So, you’ve achieved the dream of configuration management—an infrastructure full of properly configured servers that you never need to log into to manage.

But what if you want to? There are times when you may like to log into all of the members of a particular role and run a command. Fortunately, there are tools to make this task easier. These tools run the gamut from simple “run ssh in a for loop” scripts, to sophisticated remote scripting systems, like Capistrano.

Ideally, such a tool should integrate with your configuration management system. You want to be able to log into groups of servers by role, or even by service. This may require some scripting on your part to convert your role and service definitions into a format the tool understands. Alternately, the tool may be provided as a component of your configuration management or monitoring system (such as the gexec utility provided with Ganglia).

You can run commands on multiple servers. Should you?

In general, a good use for these utilities is to gather ad hoc data about your systems—perhaps information you’re not measuring with your trending tools. They’re also useful for debugging and forensics. The rule of thumb should be: If it’s not something that you should be collecting as a metric, and it won’t affect server state, it’s OK.

When is it a bad idea? You should hesitate any time it would be more appropriate to use configuration management. There’s always the possibility of forgetting ad hoc changes you made. You will regret forgetting.

Example 2: Multiple Data Centers

Eventually, you'll want to attack the greatest single point of failure of them all—the data center. You want to be able to continue to serve traffic even if your data center experiences a catastrophic power failure or other calamity. When you expand your infrastructure into multiple physical sites, you begin to realize the gains of automation on an even larger scale.

Bringing up another data center can look like a logistical nightmare on paper. It took you months or years to get your current systems in place. How will you be able to rebuild them in another location quickly? Automated deployment can make the prospect of bringing up an entire facility from bare metal much less daunting.

Rather than replicate each system in the original site on a host-by-host basis, the process unfolds as such:

- Set up management hosts in the new data center. The base installs may be manual, but the configuration is not—your management host configurations should be in configuration management as well!
- Tweak the base configurations to suit the new environment. For example, DNS configuration and routing information will differ.
- Allocate roles to the new machines on the boot server (or in inventory management).
- Boot the hosts and allow them to install and configure themselves.

To simplify synchronization of settings between data centers, it's best to keep all data center-specific configurations in a separate service or set of services. This allows you to attain maximum reuse out of your service definitions.

Summary

Knowing how much hardware you need does little good if you can't get that hardware into service quickly. Automating your infrastructure with tools like configuration management and automated installation, ensures your deployment processes are efficient and repeatable. Automation converts system administration tasks from one-off efforts into reusable building blocks.

---

[1] A substantial part of this chapter was written by my colleague Kevin Murphy at Flickr.

Appendix A. Virtualization and Cloud Computing

TWO OF THE GOALS OF CAPACITY PLANNING ARE TO EMPLOY THE RESOURCES YOU HAVE ON HAND IN THE most efficient manner, and to predict future needs based on the patterns of current use. For those well-defined workloads, you can get pretty close to utilizing most of the

hardware resources for each class of server you have, such as databases, web servers, and storage devices. Unfortunately, web application workloads are rarely (if ever) perfectly aligned with the available hardware resources.
In those circumstances, you end up with inefficiencies in your capacity. For example, if you know your database's specific ceiling (limit) is determined by its memory or disk usage, but meanwhile it uses very little CPU, then there's no reason to buy servers with two quad-core CPUs. That resource (and investment) will simply be wasted unless you direct the server to work on other CPU-intensive tasks. Even buying a single CPU may be overkill. But often, that's all that's available, so you end up with idle resources.

It's the continual need to balance correct resources to workload demand that makes capacity planning so important, and in recent years some technologies and approaches have emerged that render this balance easier to manage, with ever-finer granularity.

Server *virtualization* and *cloud computing* are two such approaches, and it's worth exploring what they mean in the context of capacity planning.

Virtualization

There are many definitions of virtualization. In general, virtualization is the abstraction of computing resources at various levels of a computer. Hardware, application, and operating system levels are some of the few places in which this abstraction can take place, but in the context of growing web operations, virtualization is generally used to describe OS abstraction, otherwise known as server virtualization.

An example of this is the Xen virtual machine monitor, or VMWare's ESX server, where a bottom-level OS functions with guest operating systems running on top of it. The bottom-level OS, known as the *hypervisor*, can be thought of as the brains of the virtualization. It allows the guest operating systems to share resources and easily be created, destroyed, or migrated to other hosts.
Entire books are written on the topic of virtualization. As it relates to capacity planning, virtualization allows for more granular control of how resources are used at the *bare metal* level. Figure A-1 illustrates this concept.

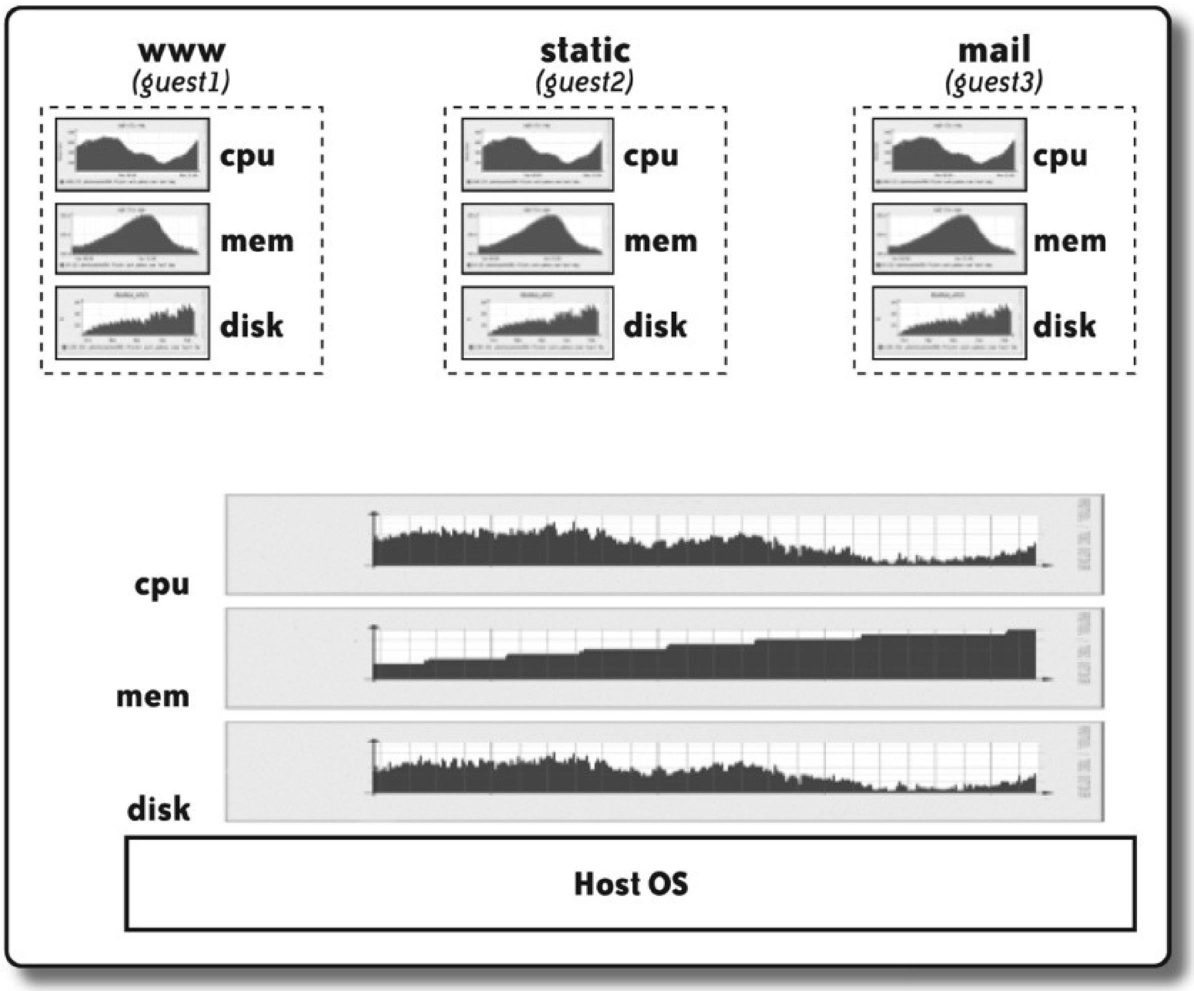


Figure A-1. Virtual servers running on bare-metal hardware

Figure A-1 shows multiple guest operating systems running on the same server. There are many advantages to employing this abstraction:

Efficient use of resources

There’s no reason to waste an entire server to run small tasks like corporate email. If there are spare CPU, memory, or disk resources, you can pile on other services to that resource to make better use it. Because of this, organizations use virtualization to consolidate many servers to run on a single piece of hardware.

Portability and fault tolerance

When a physical host is reaching known (or perhaps unknown) limits, or suffers a hardware failure, a guest OS (and its associated load) can be safely migrated to another host.

Development sandboxes

Because entire operating systems can be created and destroyed without harming the underlying host environment, virtualization is ideal for building multiple development environments that demand different operating systems, kernels, or system configurations. If there’s a major bug that causes the entire test-bed to explode, no problem—it can be easily recreated.

Less management overhead

Virtualization allows you to consolidate several individual servers with idle resources into fewer servers with higher resource utilization. This can translate into reduced power consumption, as well as a smaller data center footprint. Another benefit of less hardware is there are fewer parts subject to failure, such as disk drives, CPUs, and power supplies. Of course, the counterpoint to this is consolidation can increase your exposure to a single-point-of-failure (SPOF) as many services are dependent on the same physical hardware. Virtualization packages solve this potential problem by allowing virtual machines to easily migrate from server to server for disaster recovery, and for rebalancing workloads.

Virtualization essentially allows you to do more work with less hardware. These efficiencies have a tradeoff in that they can complicate measurements. Identifying which resource is virtual usage, and which is physical can be confusing, as the abstraction layer introduces another level of metric collection and measurement.

One additional advantage to virtualization is you can separate out application ceilings on a role-by-role basis, even when you are only running on a single physical server. For example, let’s say you’re consolidating email, backup, and logging services onto a single server. You may allocate more memory to the logging services for buffering the log writes to disk, and you may allocate more disk space to the backup application so it has room to grow.

As long as you can keep track of the virtual and the physical, the capacity planning process is roughly the same. Consider your physical servers as generic containers in which you can run a limited number of virtual servers.

Cloud Computing

The concept of packaging up computing resources (computation, storage) into rentable units, like power and telephone utilities, isn’t a new one. Virtualization technologies have spawned an entire industry of computing “utility” providers, who leverage the efficiencies inherent in virtualization to build what are known as *clouds*. Cloud service providers then make those resources available on a cost-per-usage basis via an API, or other means. Since cloud computing and storage essentially takes some of the infrastructure deployment and management out of the hands of the developer, using cloud infrastructure can be an attractive alternative to running your own servers. But as with virtualization, you lose some of the ability to monitor and precisely measure your usage.

Computing Resource Evolutions

No matter how you look at it, your website needs computing and storage resources. Somewhere—whether in your direct and total control or not—a server will need to respond to client requests for data, and those requests may need some amount of computation and data retrieval.

Virtualization has been around almost as long as computing. At one time, computers were seen as equipment only managed by large financial, educational, or research institutions. Since computers were extremely expensive, IBM and other manufacturers built large-scale minicomputers and mainframes to handle processing for multiple users at once, utilizing many of the virtualization concepts still in use today. Users would be granted slices of computation time from mainframe machines, accessing them from *thin*, or *dumb*, terminals. Users submitted jobs whose computation contended for resources. The centralized system was managed via queues, virtual operating systems, and system accounting that governed resource allocation. All of the heavy lifting of computation was handled by the mainframe and its operators, and was largely invisible to the end users. The design of these systems was largely driven by security and reliability, so considerable effort was applied to containing user environments and data redundancy. Figure A-2 illustrates the client-sever structure in a mainframe environment.

Mixed Definitions

Grid computing, cloud computing and services, managed and virtual hosting, utility services: as with many emerging technologies, terminology can sometimes be confusing—especially when they become popular. The concept of cloud computing is not immune from appellation bewilderment. Vendors, marketers, consultants, and service providers stretch the term to include nearly any scenario wherein a generic and scalable infrastructure is made available to many users on a pay-per-resource basis.

Even the marketing for various cloud computing providers acknowledges this confusion. Some of the more broad categories for these are:

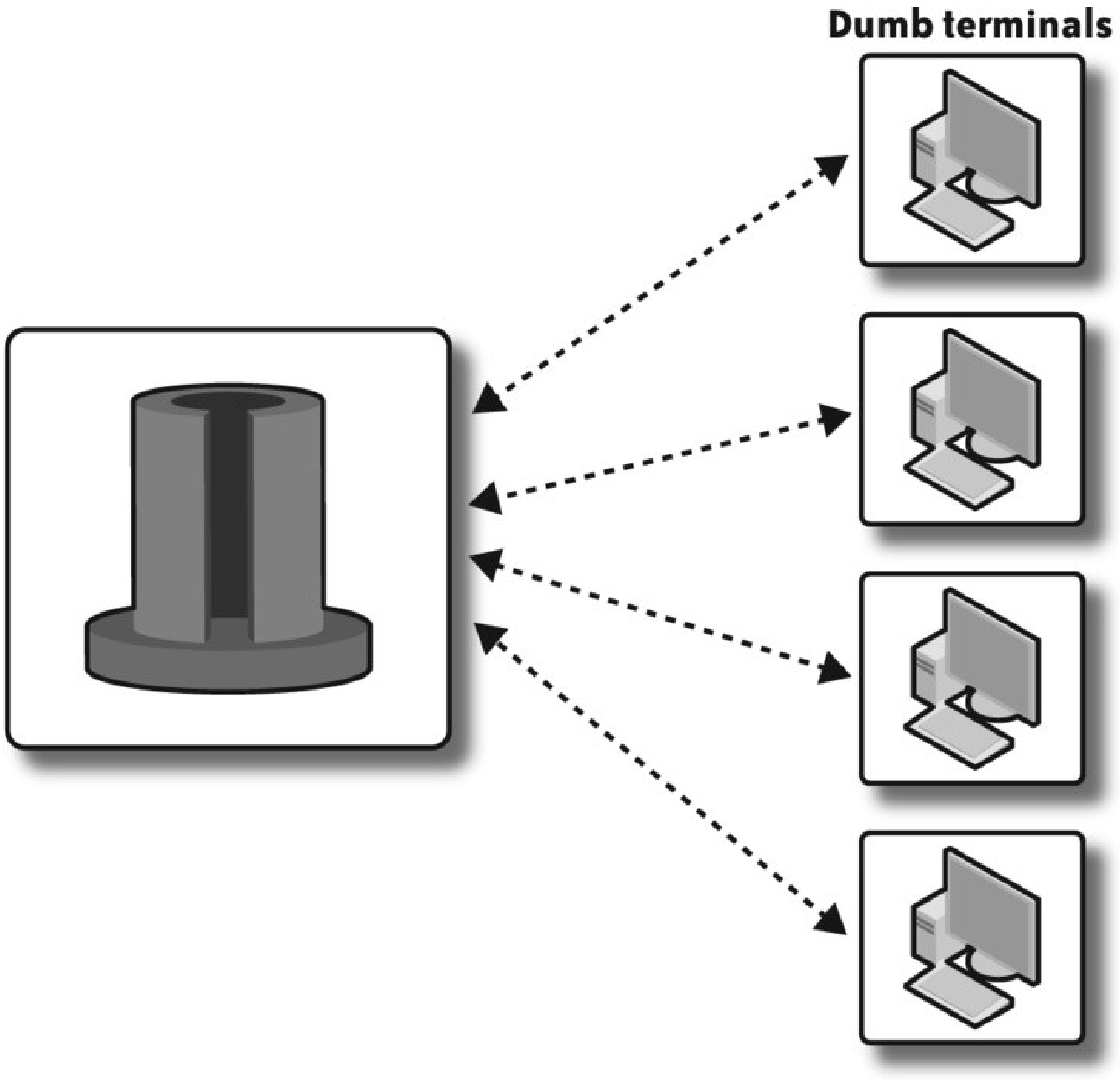


Figure A-2. Mainframe computing and the client-server paradigm

Cloud infrastructure

These are services such as Amazon Web Services and FlexiScale. They provide the lower-level elements that most resemble a managed hosting environment, with API access to control many of the services, including the ability to increase or decrease the capacity. Computation and storage are currently the main offerings.

Platform clouds

Google’s AppEngine is an example of a service that, within certain constraints, allows customers to write code that runs in a specialized environment, abstracted away from the underlying operating system and hardware resources. Web applications have access to a non-relational database that will expand or contract as usage dictates, up to prescribed limits.

Application clouds

> Salesforce.com's browser-based hosted applications are specific applications written by a vendor for specific purposes, and allow API access to applications built upon them. This is also known as SaaS, or "software-as-a-service."

For the most part, growing web applications have been looking to the first category for solutions to scaling their backend infrastructure, but as I said, this is an emerging and evolving technology. The levels of abstraction, development environments, and deployment constraints for each type of cloud computing will be right for some applications, but not for others.

There's been enough history in the cloud infrastructure category that a number of businesses have moved some of their mission-critical operations to this model with success. We'll look at some example cases a little later.

Cloud infrastructure is in many ways the next step in the evolution of the computing paradigm, which began with the mainframe. As shown in Figure A-3, cloud providers build, manage, and maintain all the physical hardware, organized in a virtualized cloud, while the customers of those computation and storage services can be both individuals and businesses.

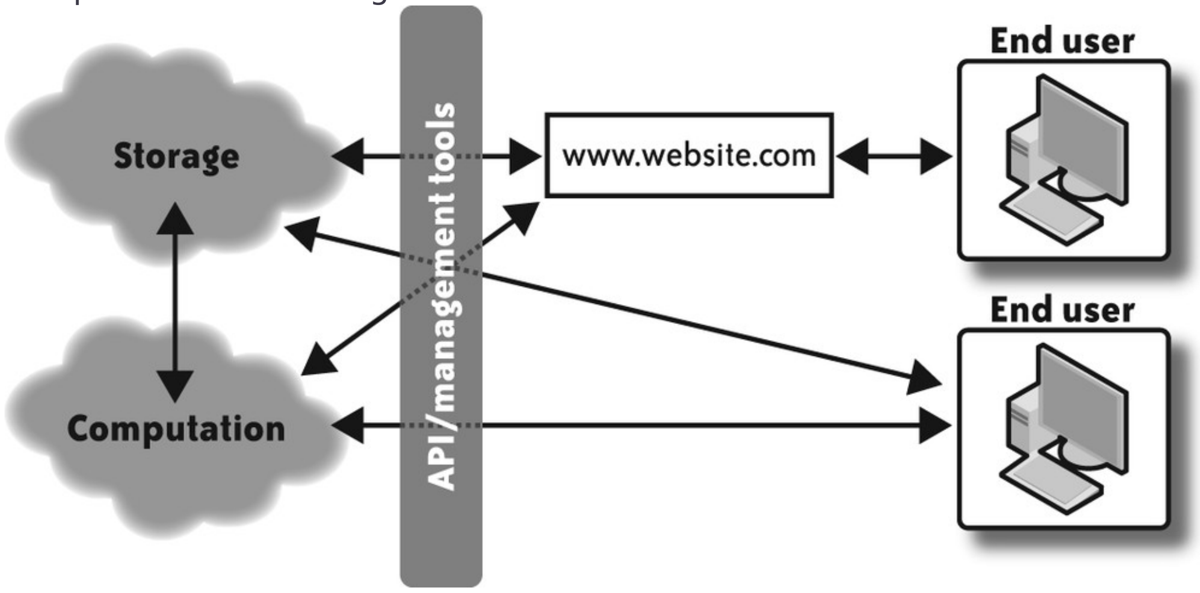


Figure A-3. Basic cloud infrastructure

The underlying architectures behind cloud computing is essentially virtualization expanded on a large scale. While there are varying types of cloud infrastructures, the prevailing variety are *compute instances* and *cloud storage*.

Compute instances are essentially virtual OS environments in which you can run your application's code. These instances can be built, torn down, and rebuilt at the customer's request. They may or may not have any persistent storage of their own to work with, so deleting an instance may wipe out the data you've written there.

On the other hand, cloud storage is basically a place to store data, and customers are billed a monthly usage rate along with transfer charges for reading and writing to the storage.

Farms of different classes of servers comprise each tier of cloud service, whether it be pure storage (in the case of a distributed file system), or a varying clusters of compute nodes that are built to house virtualized guest instances of the customer’s operating system.

Cloud Capacity

One of the most touted advantages of the cloud-computing paradigm is the reduction of hardware deployment and installation times. Indeed, a server that takes as little as 20 minutes to deploy with your automated installation and configuration process may take a minute or less with a cloud service provider. As far as planning goes, cloud storage and compute instances should be viewed as just another type of resource at your disposal. Just like a single server, for each instance of cloud-based computing, you have some amount of:

- CPU
- RAM
- Disk (persistent or non-persistent)
- Network transfer (in and out)

Each cloud resource still has its ceilings and costs, just as with your existing infrastructure. In many ways, the capacity planning process is exactly the same:

- Measure what you have consumed already (number of instances, CPU, or storage)
- Find your ceilings (when do you need to launch/tear down a new instance?)
- Forecast based on past usage

Why make forecasts if your deployment timeline is a matter of minutes? For one, forecasting isn’t solely about staying ahead of procurement and deployment timelines.

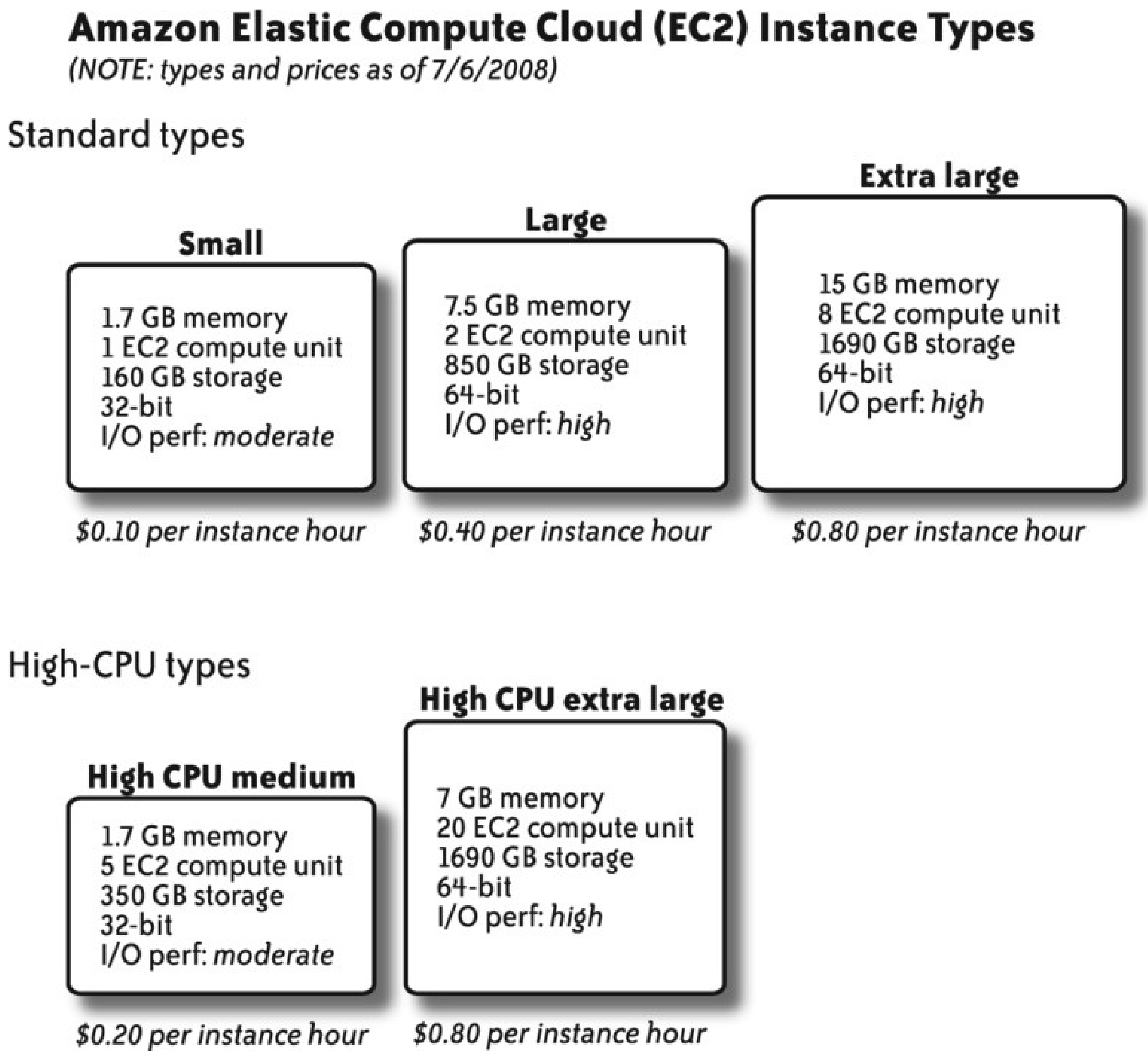
The promise of cloud computing is that you can increase capacity “on-demand” *easily*, not necessarily *automatically*. Since every instance is essentially a purchase, many cloud providers put the control of those instances in the hands of their customers, and deciding when to launch new instances (and how many) can be crucially important in the face of spiking traffic. An evolved operation that is using cloud computing might automate the process, but the automation should be tuned carefully to react not only to the load behavior of their website, but the load behavior of the cloud itself. We’ll see an example of this “load feedback” capacity tuning a little later.

As with your own servers, cloud resources cost money. One important thing to remember is the costs of cloud computing can rise surprisingly fast if you don’t pay attention. Tracking those costs will become important when you have the power to launch 100 instances at once.

Use it or lose it (your wallet)

There’s also an additional variable with cloud infrastructure that you don’t normally need to consider when deploying your own equipment: metered costs. CPU-driven cloud computing is based on the premise that you only use what you need, when you need it. Economically, it might not make sense to turn on CPU instances and simply let them run at anything less than peak capacity.

Most cloud infrastructure providers offer a “menu” of compute instance items, ranging from lower-powered CPU and memory platforms to large-scale, multi-core CPU systems with massive amounts of memory. These choices aren’t as customizable as off-the-shelf systems you own yourself, so when you determine your needs you’ll have to fit them into these more coarse-grained options. You may find many smaller-sized instances are preferable to having fewer large-sized ones, but each application is different. See Figure A-4 for an example of a cloud computing pricing menu.
As we talked about in Chapter 4, in the traditional sense of running your own hardware, the purchasing and approval process in an organization might take some time, as buying hardware is capital investment in the business. Since cloud infrastructure splits that concept into tiny chunks of investment here and there, does the approval process remain the same? Your systems administrator might not have had the authority to purchase 10 servers without approval from management. With cloud computing, does he have the authority to launch any number of instances that, over time, will equate to the same cost?


Amazon Elastic Compute Cloud (EC2) Instance Types
(NOTE: types and prices as of 7/6/2008)

Standard types

Small
1.7 GB memory
1 EC2 compute unit
160 GB storage
32-bit
I/O perf: *moderate*
$0.10 per instance hour

Large
7.5 GB memory
2 EC2 compute unit
850 GB storage
64-bit
I/O perf: *high*
$0.40 per instance hour

Extra large
15 GB memory
8 EC2 compute unit
1690 GB storage
64-bit
I/O perf: *high*
$0.80 per instance hour

High-CPU types

High CPU medium
1.7 GB memory
5 EC2 compute unit
350 GB storage
32-bit
I/O perf: *moderate*
$0.20 per instance hour

High CPU extra large
7 GB memory
20 EC2 compute unit
1690 GB storage
64-bit
I/O perf: *high*
$0.80 per instance hour

Figure A-4. Amazon EC2 menu: choices and prices

The point here is that tracking costs is another reason to pay attention and measure your cloud usage in the same way you would your own resources.

Measuring the clouds

When you operate your own infrastructure, you have many options for measuring your current capacity. Metric collection tools for historical data and trending, event monitoring for critical threshold altering, and of course OS-level tools for ad-hoc troubleshooting; these are among the tools we’ve talked about in this book.

Because cloud infrastructure is in many ways a “black box,” and because most cloud service providers offer a relatively limited menu of resource choices, the importance of effective measurement becomes even clearer. Even though deployment timelines can shrink significantly with cloud services, you should still be paying attention to the rest of the capacity planning process. Finding your ceilings is still a necessity, and deploying new instances ahead of growth still follows that.

In the case of storage consumption, you can offload the forecasting to your cloud provider; it’s their job to stay ahead of that curve. However, in the case of compute instances, the amount of actual “work” you’ll get out of them is going to depend largely on the size/class of the instance, and your application’s resource usage (taking into account, of course, the size/class of the instance).

Outside of the normal reasons for measuring your cloud’s capacity, there are other reasons for setting metric collection and event notification in place:

- The performance of any of your instances and your storage may vary as the cloud provider shuffles capacity around to rebalance their architecture for increasing load.
- Even if your cloud provider has SLAs in place, knowing when your instances or storage has failed is obviously paramount. A good rule of thumb: trust, but verify on your own, that the infrastructure is available.
- The rapid deployment of new capacity can happen quickly, but not instantaneously. It’s worth measuring how long it can take to launch a new instance.
- While cloud storage may be auto-scaling for disk consumption, there may not be any guarantees regarding how fast you can retrieve data from that storage. Just as in the disk I/O-bound database in Chapter 3, storage throughput can be a factor in your capacity, and should be measured. The same goes for “local” storage I/O that any of your compute nodes experiences.

With most cloud providers, you may not have any clarity on how your allocated instances run (or sometimes where). They may be running by themselves on brand new hardware. They may be sharing hardware resources alongside other virtualized instances that have their own fluctuating capacity needs. They may be located on a network switch whose usage is close to its limits. Your instances may or may not have consistent latency in storing or retrieving data from its persistent storage. You simply don’t know any of the physical details under the covers of your cloud provider’s interface.

This lack of transparency brings some advantages. It allows for the cloud provider to make unfettered changes to their operation (which can enable the rapid provisioning of new instances) and also relieves the customer of concerns about the nitty-gritty details involved in scaling the capacity. But it also means you don’t have all the details you may wish, especially when you’re accustomed to having every detail at your fingertips.

Cloud Case Studies

Should you use cloud services for your growing website? The short answer: “it depends.” The longer answer is, cloud infrastructure, like every technology, has its pros and cons. On the one hand, ridding yourself of deployment and provisioning hassles may be benefit enough for you to switch to clouds. On the other hand, you give up some degree of control that you may be used to.

It’s too early in the history of cloud computing, and the variety of offerings is too great to set up heuristics and formal guidelines for when to use it. Anecdotes and case studies, however, can help you decide whether you’re in the right class of user and what factors to consider. As I mentioned earlier, some businesses have been able to migrate some of their mission-critical services to clouds with varying success. To illustrate this, I’ve included some cases below. In preparation for this book, I interviewed a number of organizations that have either direct experience using cloud infrastructure or have evaluated them as part of the process of building out new capacity.

Each one of these use cases highlights how drastically different the needs, concerns, and benefits can be for each organization when using cloud infrastructure.

Cloud Use Case: Anonymous Desktop Software Company

A well-established desktop software company investigated using cloud infrastructure to store and host digital media its users would process and upload from its product. Being a desktop software company, it didn’t have a large online presence until this product was launched. Nor did it have the necessary operational staff to manage the clusters of servers needed to provide the hosting. The company considered using cloud infrastructure, hoping to offload some of the deployment and management overhead that comes with a 24/7 online product. It evaluated the idea at the technical and business levels, looking at the pros and cons.

In the end, the company decided to run their own infrastructure, for several reasons.

Lack of suitable SLAs

At the time, SLAs were only available in very limited terms with most of the established cloud service providers. Being relatively new to the online world, the company didn’t have high confidence in trusting a third-party (even a well-known cloud provider) with the reliability and performance of its customer’s data, so the lack of a comprehensive SLA played a significant role in their decision.

Legal concerns of user data

A concern was raised about who actually owns the user's data while it's being hosted on the cloud provider's servers. Did the cloud provider have any rights to the data since it was physically located on its servers? Were there restrictions put in place that respected national or international borders with regard to hosting? Legal issues involving user data, privacy, and reliability are always fraught with what-if scenarios, and also played a role in this particular company's decision.

Cost

As it was an already established business, it had the capital to host the data itself, so the economics of using cloud services were different from that of a startup. Once the company factored in what it would cost to operate its own data center with the forecasts of the online feature's adoption, it decided the investment was going to be worth it in the long-term, based on the cost histories of cloud providers at that point in time, as well as the company's own Total Cost of Ownership (TCO) calculations.

Control and confidence

Another main reason for the company not to use cloud infrastructure was a non-technical one. Developing it themselves allowed the company to have familiarity with what it was building. If the system failed, the company felt empowered to fix it in whatever manner necessary. It also didn't want to redesign its application in order to utilize the cloud resources via the APIs. Finally, the company felt more comfortable that it could move to the cloud sometime in the future, once the feature's usage patterns emerged.

Cloud Use Case: WordPress.com

WordPress.com hosts over 2 million blogs (as of this writing), and gets upward of 30 million pageviews a day. The company's servers are located in three different data centers which all replicate their data. For some time, this made uploading media (video, audio, photos) difficult for the user simply because the company didn't have a good handle on deploying new storage, so it looked into using Amazon's Simple Storage Service (S3) as a backup/data recovery solution.
As WordPress became familiar with the service, it started to use S3 for its primary storage. The reasoning behind using cloud storage was not economics; at the time of this writing, S3 costs are actually three to four times more than if it bought and managed its own storage. The advantage it was looking for was the ease of deployment and management. Not having to keep ahead of storage usage meant they could focus on the

other parts of the infrastructure and site features. WordPress intended to use S3 essentially as a near-infinite storage bin.

Because Amazon Web Services (AWS) charges for transferring data in and out of its S3 cloud, WordPress reverse-proxy caches the content it takes out of S3 to its own servers (see Figure A-5).

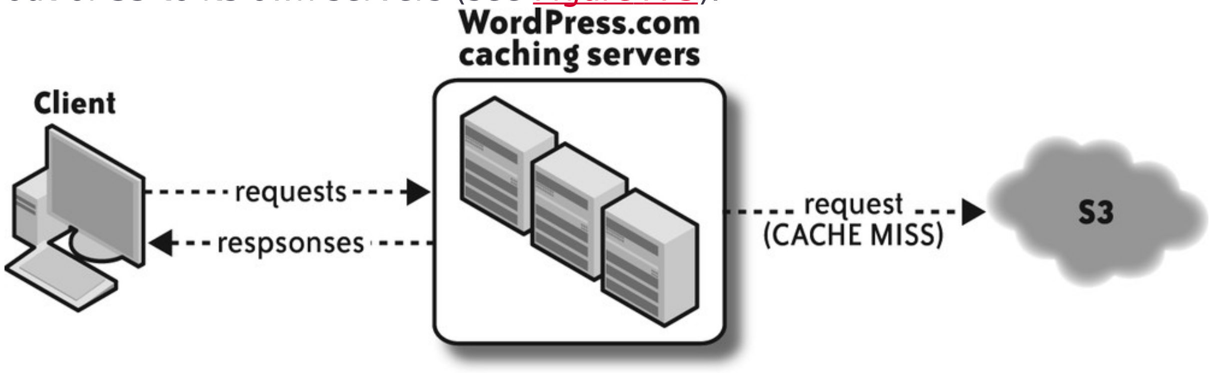


Figure A-5. WordPress.com reverse-proxy caching of Amazon of S3 storage

By caching frequently requested objects (or objects deemed to be "hot" enough to cache), WordPress uses S3 in the most efficient way possible, avoiding unnecessary transfer costs and serving content quickly via its own servers.

Caching in this manner allows it to use S3 as a near-infinite storage resource. It doesn't need to worry about the limits and overhead involved with incrementally deploying new storage itself.

Does this mean WordPress has ceased to do capacity planning? Far from it. It now has caching systems (and databases and web servers) that need to be scaled, but this is acceptable, as storage was its biggest pain point. WordPress was happy to redesign how it uses storage in order to avoid worrying about it again.

Cloud Use Case: Anonymous News Aggregation Site

An early-stage startup decided to use cloud infrastructure to run its development environment as well as its limited beta website. The site is designed to crawl the Web and index well-known news sources. The company would then process those documents with some natural-language algorithms in order to uncover articles in which its users would be interested. It used compute instances to do the crawling and would store both the raw data and the processed data in cloud storage.

While developing the site, the company noticed performance in its cloud-hosted application would vary, with no emerging pattern. Sometimes, it would bring up a new compute instance and notice the processing speed would vary from instance to instance, or would sometimes pause inexplicably in the middle of a computation.

In addition, since the company’s processing was CPU intensive, it would launch many smaller instances from the cloud provider’s menu of choices. As the application grew and the company started using more and more of the smaller instances, they considered upgrading to the next tier of the compute service for more CPU horsepower. While it was more attractive to launch fewer “large” CPU instances to have more processing power on hand, the pricing menu in place at the time made it prohibitively expensive when compared to the smaller instances. This meant it would be forced to continue running many small instances and accept the degraded memory and local I/O subsystems performance. This wasn’t an ideal situation.

This gap in the cloud provider’s pricing menu caused the startup to conclude it should run its own systems, and the organization began investigating managed hosting and co-location facilities. The cloud provider then introduced changes to its pricing menu, offering a “medium” level that better suited the startup’s budget and capacity needs.

This case underscores that cloud infrastructure services are still evolving their businesses and still adjusting to their customer’s needs with regard to configurations and management tools.

It also reinforces the importance of monitoring. When the startup began using Nagios to monitor its instances, it was able to see and record how the instances were performing, and make much more educated decisions on whether to launch more compute instances.

Cloud Use Case: SmugMug.com

SmugMug.com is a photo-sharing website, not unlike Flickr. It manages its own web servers and databases, but it uses Amazon’s cloud infrastructure extensively for both storage (S3) and compute processing (EC2).
As of this writing, SmugMug.com used over 600 TB of space with Amazon’s S3, and offloading this storage capacity to Amazon’s cloud allows it to focus on developing new features. It started by using S3 as backup storage for its site. Once SmugMug felt comfortable with performance and reliability, it moved all primary storage to S3, even when Amazon Web Services did not have an SLA for this service (it does now). Moving storage was an easy decision for SmugMug because it saw economic benefits and also did not want to increase the size of its operations staff to manage in-house storage.
When photos are uploaded to SmugMug, they are placed in a queue that ships them off to EC2 for processing into their various sizes for use on its site as well as other photo processing actions. The same is done for

uploaded video. Once the media has been processed, it’s then stored directly to S3. This is an oversimplification of course, but during this process, SmugMug manages their cloud use in an interesting way.

Capacity feedback loops

Since SmugMug automates the transfer and processing of the videos and photos to the cloud via queues, it can control the rate at which processing jobs are sent, and how many compute instances it might need at any given time to handle the load. Every minute, SmugMug’s system looks at a number of different metrics to decide whether to turn on or tear down compute instances, based on need. Some of the metrics it looks at are:

- Number of currently pending jobs
- Priority of pending jobs
- Type of jobs (photos, video, and so on)
- Complexity of the jobs (HD video versus 1 megapixel photo)
- Time sensitivity of pending jobs
- Current load on EC2 instances
- Average time per job
- Historical load and job performance
- Length of time it’s currently taking to launch a new compute instance

By taking these metrics into account, SmugMug’s system can appropriately scale up or down the number of EC2 instances it’s using at any given time, thus making processing much more efficient than without cloud infrastructure.

SmugMug’s use case follows the tenor of Chapters Chapter 3 and Chapter 4: marrying application and system metrics to plan for capacity, and making forecasts based on that historical data.
In SmugMug’s case, it is aware of its ceilings, monitoring how close it is to reaching them, and adjusting its capacity on a tight, sliding window of time.

For SmugMug, taking into account its Total Cost of Ownership calculations, its desire to keep its team small, and its scale, it’s economically advantageous for it to use cloud infrastructure for storage and processing.

Summary

Deploying your site to cloud infrastructure can change how you view deploying capacity, and largely depends on how you intend to make efficient use of it. In the use cases above, we see both non-technical, and technical considerations, as outlined in the lists that follow.

Non-technical considerations:

- Legal concerns surrounding privacy, security, and ownership of data by a third-party.
- Confidence in the availability and performance of cloud infrastructure.
- The effect of SLAs (or lack thereof) in the context of *pieces* of your infrastructure.
- Levels of comfort with a still-emerging technology platform.

Technical considerations:

- Redesigning their own application to make the most efficient use of cloud resources. Architectures that avoid transfer costs when possible and deploying compute instances only when you need them are very common practices.
- Not knowing where your data physically resides. This forces developers to think about their application (and the management of their application) at a higher-level. Expecting that compute instances can stall, disappear, or migrate requires redundancy to be built in.

Regardless of how organizations decide to use cloud infrastructure, its effect on capacity planning can be significant. Wordpress.com is paying *more* for their storage than it did prior to migrating its data storage, but is comfortable with that. SmugMug.com is paying *less* for Amazon S3 than it would if it were managing its own storage. Ultimately, there is no one-case-fits-all situation with respect to cloud infrastructure; each decision is dependent on the application and organization involved, just as with so many other technologies.

Clouds can shrink deployment timelines and provide more granular control over how you're using your capacity. These are facets of capacity management that we've discussed in the previous chapters, and should be applied to cloud infrastructure as well:

- Put capacity measurement into place—both metric collection and event notification systems—to collect and record systems and application statistics.
- Discover the current limits of your resources (utilization on compute nodes, for example) and determine how close you are to those limits.
- Use historical data not only to predict what you'll need, but to compare against what you will actually use.

Planning for growth using cloud infrastructure is an evolving area. Cloud providers have different restrictions, features, benefits, and drawbacks when compared to running your own infrastructure. But once you have a good capacity planning process, you'll be able to adapt your planning methods to take those into account.